МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

В. В. Шликов, В. А. Данілова

ВИСОКОПРОДУКТИВНІ РОЗПОДІЛЕНІ ОБЧИСЛЮВАЛЬНІ СИСТЕМИ

Практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 122 «Комп'ютерні науки»,
спеціалізацією «Інформаційні технології в біології та медицині»

Київ
КПІ ім. Ігоря Сікорського
2018

Рецензенти: *Лебедєв В.О., д.т.н.*, професор
Соломін А.В., к ф.-м.н, доцент

Відповідальний редактор *Зубчук В.І., к.т.н.*, доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 1 від 27.09.2018 р.) за поданням Вченої ради факультет біомедичної інженерії (протокол № 1 від 07.09.2018 р.)

Електронне мережне навчальне видання

Шликов Владислав Валентинович, канд. техн. наук, доцент,
Данілова Валентина Анатоліївна

ВИСОКОПРОДУКТИВНІ РОЗПОДІЛЕНІ ОБЧИСЛЮВАЛЬНІ СИСТЕМИ

Високопродуктивні розподілені обчислювальні системи: Практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 122 «Комп'ютерні науки», спеціалізації «Інформаційні технології в біології та медицині» / В.В. Шликов, В.А. Данілова; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: 1,6 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2018. – 108 с.

Практикум з дисципліни «Високопродуктивні розподілені обчислювальні системи» містить практичні заняття та лабораторні роботи, що виконуються за допомогою програмного забезпечення в середовищі програмування Microsoft Visual Studio 2010 на мові C# і C++ та в середовищі програмування Code Composer Studio для цифрових мікропроцесорів сімейства Texas Instruments DSK64xx. Навчальне видання рекомендовано для вищих навчальних закладів України з викладанням спеціальності 122 «Комп'ютерні науки».

ЗМІСТ

Вступ

Метою практичних занять та лабораторних робіт з дисципліни «Високопродуктивні розподілені операційні системи», виконуваних за допомогою програмного забезпечення в середовищі програмування Microsoft Visual Studio 2010 на мові C# і C++ та цифрових мікропроцесорів сімейства Texas Instruments DSK6400 в середовищі програмування мікропроцесорів Texas Instruments (Code Composer Studio) є:

– вивчення принципів побудови та програмування мультипроцесорних обчислювальних систем;

– вивчення принципів побудови систем з масовою паралельною обробкою і кластерних обчислювальних систем;

– розробка програмного забезпечення для роботи з неіменованими каналами та основними механізмами реалізації технологій "клієнт-сервер";

– вивчення принципів побудови операційних систем (прозорість, гнучкість, надійність, ефективність, масштабованість).

– розробка програмного забезпечення для реалізації комунікацій в розподілених системах і синхронізації в розподілених системах.

В результаті виконання лабораторних робіт студент повинен вміти остаточно формулювати вимоги до мультипроцесорних обчислювальних систем (МОС). В процесі розробки програмного забезпечення для МОС студенту необхідно вміти застосовувати механізмами реалізації технологій "клієнт-сервер", а також програмні алгоритми на мові C# та C++, в яких використовуються паралельні процеси, що можуть взаємодіяти декількома основними способами:

– за допомогою розподілу пам'яті (оперативної або зовнішньої);

– за допомогою передачі повідомлень.

Завдання на лабораторні дослідження розраховані на 2 академічні години. Результати проведених досліджень і вимірювань повинні бути задокументовані і в кінці заняття представлені викладачу. За виконаної лабораторної роботи складається звіт, який повинен містити:

1. Титульний аркуш звіту.
2. Теоретичні відомості з теми дослідження.
3. Схеми МОС, для яких розроблено програмне забезпечення.
4. Програмний код, що ілюструє досліджені процеси.
5. Висновки за результатами роботи алгоритмів.
6. Відповіді на контрольні питання.

1. Загальні відомості з мультипроцесорних обчислювальних систем

Мультипроцесорні обчислювальні системи (МОС) мають велику гнучкість, зокрема вони можуть функціонувати як високопродуктивні однозадачні обчислювальні системи, та як багатопрограмні обчислювальні системи (ОС), що паралельно виконують паралельні задачі. Крім того, архітектура МОС дозволяє більш ефективно використовувати усі переваги сучасної мікропроцесорної технології.

У МОС кожний процесорний елемент (ПЕ) системи виконує власну програму незалежно від інших ПЕ. У той же час ПЕ системи взаємодіють один з одним. Способи такої взаємодії визначають умовне ділення ОС, що використовуються у МОС, на обчислювальні системи з розподіленою пам’яттю та системи з загальною пам’яттю.

В системах з розподіленою пам’яттю, які представляють собою слабо зв’язані багатопроцесорні системи, уся пам’ять розподілена між процесорними елементами системи, а кожний блок пам’яті доступний тільки процесору, що його обслуговує. Мережа з’єднань зв’язує всі процесорні елементи один з одним. Представниками цієї групи МОС є кластерні обчислювальні системи (Cluster System) та системи з масовим паралелізмом (MPP, Massively Parallel Processing).

В системах з загальною пам’яттю, які представляють собою сильно зв’язані системи, до загальної пам'яті даних і команд мають доступ усі ПЕ, що забезпечується за допомогою загальної шини або мережі з’єднань. До цього типу, відносяться системи з неоднорідним доступом до пам’яті (NUMA, Non-Uniform Memory Access) та симетричні мультипроцесори (SMP, Symmetric Multiprocessor).

Базова модель обчислень на МОС описується сукупністю незалежних процесів, що періодично звертаються до сумісно використовуваних даних. Характерні значення пікової продуктивності для потокових обчислень для різних типів МОС представлені на рис. 1.1.

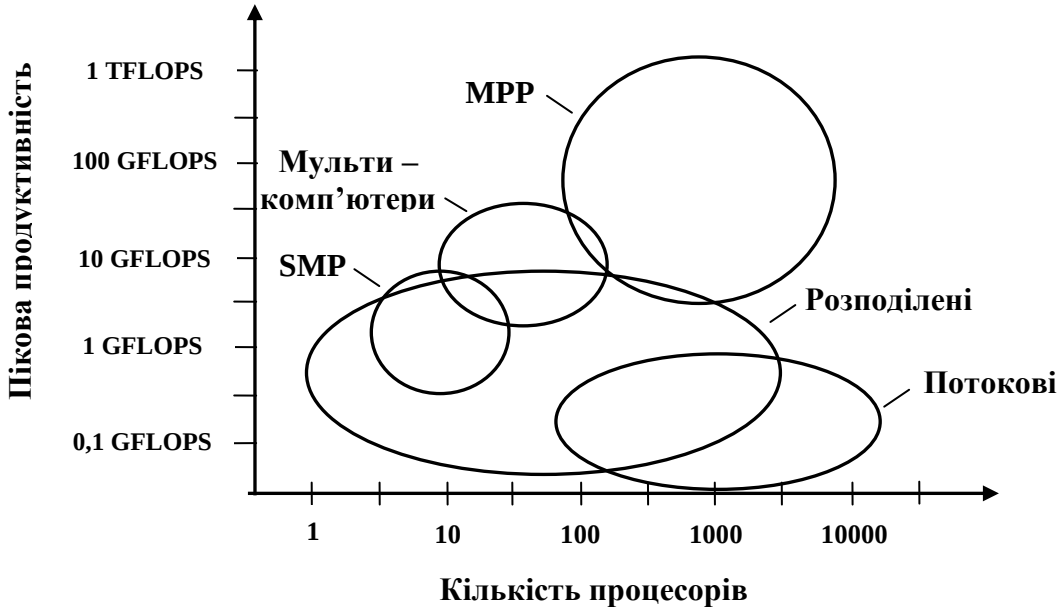


Рис. 1.1. Продуктивність МОС як функція їх типу та кількості процесорів

Для моделі обчислень на МОС існує безліч варіантів реалізації. На одному рівні забезпечення продуктивності систем знаходяться розподілені обчислення, у рамках яких програма ділиться на достатню множину паралельних задач, що складаються з великого числа підпрограм. На іншому рівні – модель потокових обчислень, де кожна логічна і арифметична функція в програмі може розглядатись як окремий процес. Такі операції очікують на надходження вхідних даних (операндів), які повинні бути передані до функцій іншими процесами. Після цього операцію може бути виконано, а результуюче значення передається наступним процесам, які виконуються при обчислені.

1.1. Симетричні мультипроцесорні системи (SMP)

Поняття симетричної мультипроцесорної обчислювальної системи, що має назву SMP-системи (Symmetric Multiprocessor), відноситься як до архітектури обчислювальної

системи, так і до операційної системи, яка відображає дану архітектурну і програмну організацію МОС. Систему SMP можна визначити як обчислювальну систему, що має наступні характеристики:

– два або більше процесорів, що мають порівняну продуктивність;

– процесори, які сумісно використовують основну пам’ять і функціонують в єдиному фізичному і віртуальному адресному просторі;

– процесори, які зв’язані між собою за допомогою шини або за іншою схемою, що забезпечують однаковий час доступу до пам’яті будь-якого з них;

– процесори, які розділяють доступ до пристроїв вводу-виводу через сумісні канали, або через різні канали, що забезпечують доступ до потрібного зовнішнього пристрою;

– процесори, які здатні виконувати однакові функції та забезпечують «симетричну» організацію системи;

– будь-який з процесорів МОС, який може обслуговувати зовнішні переривання;

– обчислювальна система, яка управляється інтегрованою у процесор операційною системою, що координує і організовує взаємодію між програмами та процесорами на рівні функцій і задач, файлів і елементів даних.

SMP-системи у порівнянні з однопроцесорними схемами МОС мають переваги по наступним показникам:

1. Продуктивність. Якщо окремі частини задачі, що розбита на декілька частин, можуть виконуватись паралельно, то застосування множини процесорів дає виграш в продуктивності відносно однопроцесорної системи того ж типу.

2. Готовність. Якщо будь який з процесорів мультипроцесорної системи здатний виконувати ті ж функції, що і інші, то в SMP-системі відмова одного з компонентів не приводить до відмови системи в цілому.

3. Розширюваність. Загальна продуктивність SMP-системи може бути збільшена додаванням у систему додаткових процесорів.

4. Масштабованість. Якщо варіювати число процесорів в SMP-системі, то можна створювати системи різної вартості та продуктивності.

На рис. 1.2 в загальному вигляді представлено архітектуру симетричної мультипроцесорної обчислювальної системи. Типова архітектура SMP-системи містить від

двох до 32 ідентичних процесорів, в якості яких зазвичай використовують однопроцесорні RISC-процесори, наприклад DEC Alpha, Sun SPARC, MIPS або HP PA-RISC. В останній час для оснащення SMP-систем використовуються також CISC-процесори, наприклад Pentium.

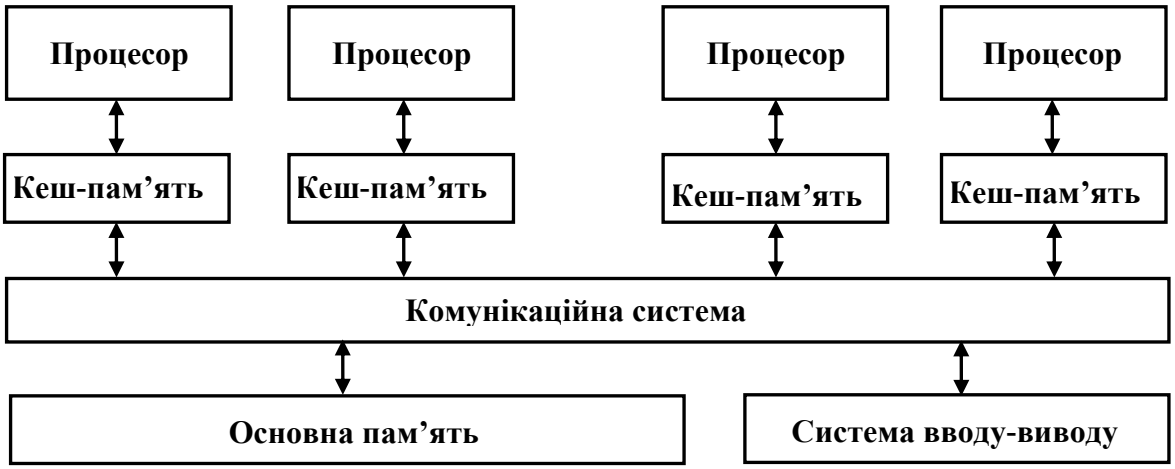


Рис. 1.2. Організація симетричної мультипроцесорної системи

Кожний процесор у мультипроцесорній системі забезпечений локальною кеш-пам'яттю, яка на логічному рівні складається з кеш-пам'яті першого (L1) та другого (L2) рівнів. Узгодженість роботи кеш-пам'яті усіх процесорів в системі забезпечується апаратними засобами. Проблема когерентності в деяких SMP-системах знімається за рахунок використання загальної кеш-пам'яті. Однак, цей прийом економічно і технічно виправданий тільки у випадку, якщо кількість процесорів у системі не перевищує чотирьох. Застосування загальної кеш-пам'яті супроводжується зменшенням швидкодії кеш-пам'яті та збільшенням вартості МОС.

Усі процесори симетричної обчислювальної системи мають рівноправний доступ до основної пам'яті та системи вводу-виводу, що фізично розділяються. Така можливість забезпечується можливостями комунікаційної системи. Усі процесори симетричної МОС взаємодіють між собою через основну пам'ять. В деяких SMP-системах передбачається можливість прямого обміну сигналами між процесорами.

Пам'ять SMP-систем будується по модульному принципу і організована таким чином, щоб забезпечити одночасне звертання до різних її модулів (банків). В деяких

конфігураціях МОС в доповнення до апаратних ресурсів, які використовуються сумісно, кожний процесор має також власні канали вводу-виводу та основну пам’ять.

1.2. Системи з масовою паралельною обробкою (MPP)

Основною ознакою обчислювальної системи з масовою паралельною обробкою (MPP, Massively Parallel Processing) є наявність великої кількості процесорів n. Строгої межі при визначені типу обчислювальної системи не існує, але при n ≥ 128 вважається, що це MPP-система. Узагальнена структура МОС з масовою паралельною обробкою представлена на рис. 1.3.

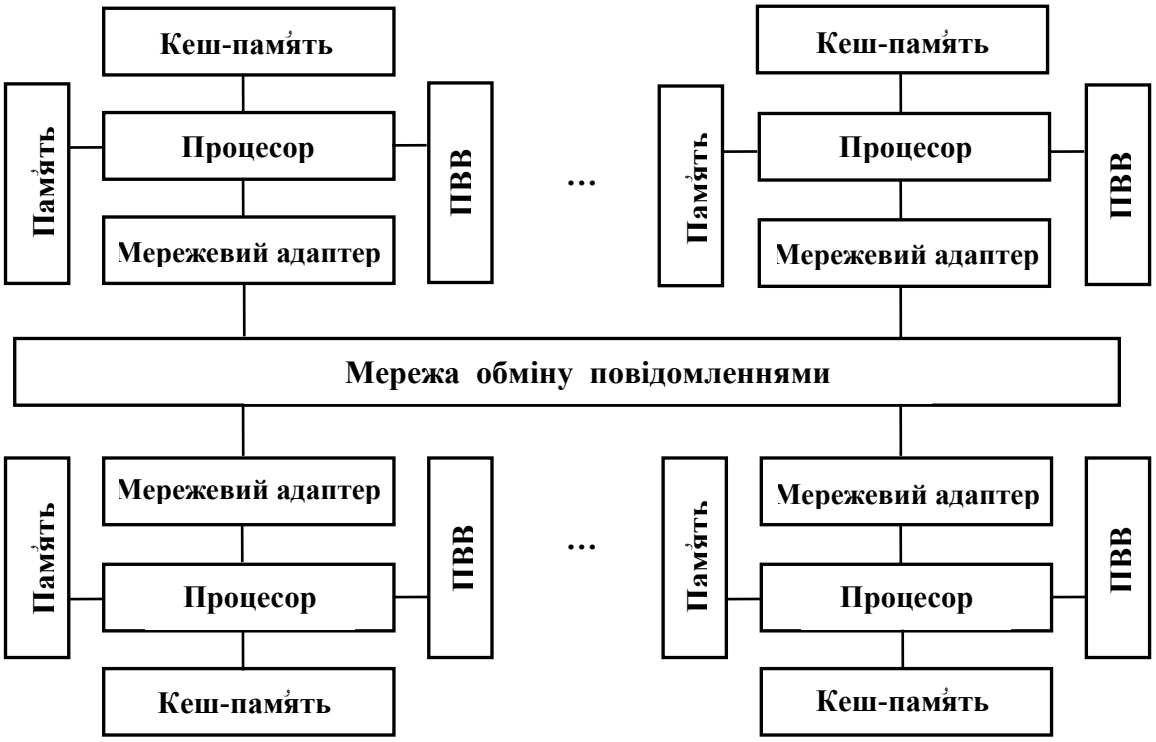


Рис. 1.3. Структура обчислювальної системи з масовою паралельною обробкою

Головні архітектурні особливості, за якими обчислювальну систему відносять до класу MPP-систем, можна сформулювати наступним чином:

– стандартні CISC- або RISC-мікропроцесори;

– фізично розподілена пам'ять;

– мережа з’єднань, що має високу пропускну здатністю та малі часові затримки;

– масштабованість, що передбачає можливість варіювання числом процесорів;

– асинхронна MIMD-система з пересилкою повідомлень;

– програма, яка реалізує множину процесів, що мають окремі адресні простори.

Основними причинами використання МОС-систем з масовою паралельною обробкою є зменшення вартості ОС та необхідність побудови ОС, що мають велику продуктивність обробки даних.

Характерна риса існуючих МОС з масовою паралельною обробкою – наявність єдиного управляючого пристрою (процесора), що розподіляє завдання (процеси) між усіма процесорами, які частіше за все є однаковими (взаємозамінними), які належать одному або декільком типам. Схема взаємодії процесорів в загальних рисах достатньо проста:

– управляючий (центральний) пристрій формує чергу завдань, кожному з яких призначається необхідний рівень пріоритету;

– по мірі звільнення процесорів і підлеглих до них пристроїв їм передаються завдання з черги;

– підлеглі пристрої оповіщають центральний процесор системи про хід виконання завдань і завершення виконання або потребу в додаткових ресурсах;

– центральний пристрій контролює роботу підлеглих процесорів і виявлення нештатних ситуацій, зокрема переривання виконання завдань у випадку більш пріоритетної задачі і т. ін.

У деякому наближенні вважається, що на центральному процесорі виконується ядро операційної системи, що має планувальник завдань, а на підлеглих – додатки. Підлеглість між процесорами може бути реалізована на апаратному, або на програмному рівні.

Завдяки функції масштабованості, на сьогодні МРР-системи є лідерами по продуктивності, що досягається. З іншого боку, розпаралелювання задач в МРР-системах є складним рішенням. Ефективність розпаралелювання в МРР-системах сильно залежить від деталей архітектури, наприклад топології схем при з’єднанні процесорних вузлів. Для синхронізації процесів, що виконуються системою паралельно, необхідно забезпечити обмін повідомленнями, які повинні доходити від будь-якого керуючого вузла до будь-якого іншого вузла системи. Час передачі даних від вузла до вузла значно залежить від стартової затримки

та швидкості передачі. Продуктивність усіх процесорів системи зазвичай набагато більше пропускної здатності каналів зв'язку, тому в МРР-системах інфраструктура каналів зв'язку є найбільш проблемною.

Слабким місцем МРР-системи є центральний управляючий пристрій (ЦУП). При виході ЦУП зі строю вся система стає не працеспроможною. Підвищення надійності ЦУП здійснюється за рахунок дублювання апаратури або її спрощення.

Незважаючи на складності в програмно-апаратній реалізації, сфера застосування ОС з масовим паралелізмом постійно розширюється. Різні системи цього типу експлуатуються у багатьох сучасних суперкомп'ютерних центрах світу.

1.3. Кластерні обчислювальні системи

Один з сучасних напрямів в області створення мікропроцесорних обчислювальних систем – це кластерізація. По коефіцієнту готовності та продуктивності кластерізація являє собою альтернативу сучасним симетричним МОС. Обчислювальний кластер представляє собою групу взаємоз'єднаних обчислювальних систем (вузлів), що функціонують сумісно, складаючи єдиний обчислювальний ресурс і створюючи єдину віртуальну обчислювальну машину. У якості вузла кластера може бути використана однопроцесорна ЕОМ, а також ОС типу МРР або SMP. Важливою ознакою кластерної МОС є те, що кожний вузол системи може функціонувати самостійно і окремо від кластера. Відносно до архітектури МОС сутність кластерних обчислень зводиться до принципу об'єднання декількох вузлів з високошвидкісною мережею. Для опису такої архітектури МОС, крім терміну «кластерні обчислення», часто використовуються такі загальновідомі назви, як: паралельні обчислення на базі мереж, кластер робочих станцій, гіперобчислення, ультра-обчислення.

У якості вузлів кластерних МОС можуть використовуватись як однакові ОС (гомогенні кластери), так і змішані ОС (гетерогенні кластери). За своєю архітектурою кластерна МОС є слабко зв'язаною обчислювальною системою.

На рівні апаратного забезпечення кластер представляє собою сукупність незалежних і самостійно функціонуючих обчислювальних систем, які об'єднані в єдину мережу. При з'єднанні ЕОМ у кластер практично завжди підтримуються прямі зв'язки між елементами системи. Кластерна архітектура може бути простою, що ґрунтуються на апаратурі Ethernet,

або складною з використанням високошвидкісних мереж з пропускною здатністю у сотні мегабайтів в секунду.

Вузли кластера у системі можуть контролювати працездатність один одного та проводити обмін специфічною інформацією, що є характерною для обчислювального кластера. Контроль працездатності кластера здійснюється за допомогою спеціального сигналу, який називають heart-beat, що означає «серцебиття». Цей сигнал передається один одному вузлами кластера, щоб підтвердити нормальне функціонування системи.

Невід'ємною частиною кластера є спеціалізоване програмне забезпечення (ПО), на яке покладається задача забезпечення безпечної роботи системи при відмові одного або декількох вузлів. Таке ПО забезпечує перерозподіл обчислювального навантаження у системі при відмові одного або декількох вузлів кластера, а також відновлення обчислень без втрати даних при виникненні збою у вузлі кластеру. Крім того, якщо у комп'ютерному кластері використовуються диски для зберігання інформації, кластерне ПО підтримує функціонування єдиної файлової системи.

Технічна можливість практично необмеженого нарощування числа вузлів, а також відсутність єдиної операційної системи у такій обчислювальній системі робить кластерні архітектури ефективно масштабуємими, навіть якщо на практиці кластерні системи мають сотні та тисячі вузлів.

1.4. Цифрові системи передачі даних

Цифрові системи передачі (СП), як правило, будуються за принципом однієї системи більш високого рівня і декілька систем нижчого рівня. Об'єднання потоків нижчого рівня проводиться блоками групового формування системи більш високого рівня. У відмінності від традиційної цифрової системи передачі, в СП на основі МОС є можливість зміни структури блоків групового формування на структуру з двох однакових половин. Перша половина блоків групового формування об'єднує перші дві системи низького рівня, друга половина – об'єднує другі дві системи низького рівня. При цьому блоки групового формування схемотехнічно та конструктивно однакові, відмінність тільки в програмному забезпеченні. Таким же чином будуються блоки групового формування і на приймальній стороні. В результаті цифрова СП на основі МОС складається з двох половинок – блоків

групового перетворення даних як на передаючій так і на приймальній стороні. При безпосередньому розташуванні системи низького рівня з системою більш високого рівня є можливість заміни послідовного потоку даних на паралельний.

Таким чином, введення даних у блоки більш високого рівня здійснюється по паралельній шині безпосередньо з блоків аналого-цифрового перетворення, за винятком ступенів перетворення паралельно - послідовної та послідовно - паралельної обробки. На приймальній стороні також є можливість заміни послідовного потоку даних на паралельний блоків СП більш високого рівня до СП більш низького рівня.

Високопродуктивною реалізацією МОС є цифровий сигнальний процесор (ЦСП) TMS320C64x (в т.ч. DSP TMS320C6455) на основі платформи TMS320C6000. Платформа C6455 розроблена на базі архітектури третього покоління VelociTI з підтримкою довгих слів інструкції (VLIW), яка розроблена Texas Instruments і робить даний ЦСП ідеальним вибором для застосування в відео і телекомунікаційному обладнанні, системах обробки зображень, медичних додатках і системах для організації відеоконференції.

Процесори TMS320C64x виготовляються за технологією 90 нм і на тактовій частоті 1 ГГц забезпечують продуктивність 8000 мільйонів інструкцій у секунду (або 8 млрд. 16-розр. множень-накопичень в секунду), що робить C6455 вигідним інструментарієм для вирішення складних завдань цифрової обробки. C6455 має операційну гнучкість високопродуктивних контролерів і обчислювальні можливості матричних процесорів. ЦСП C64x має сумісність знизу вгору по програмному коду з попередниками, які виконані на основі платформи ЦСП C6000.

Блок-схема процесору DSP TMS320C6455 представлена на рис. 1.4.

Основні характеристики і відмінні особливості процесору TMS320C6455:

1. Високопродуктивний ЦСП з фіксованою точкою (C6455)

– тривалість циклу інструкції 1,39; 1,17; 1 нс;

– тактова частота 720 МГц, 850 МГц, 1 ГГц;

– вісім 32-розрядних інструкцій / цикл;

– 5760, 6800, 8000 мільйонів інструкцій у секунду;

– 5760, 6800, 8000 мільйонів множень-накопичень в секунду (16 розрядів);

– комерційний температурний діапазон (0°C ... +90°C).

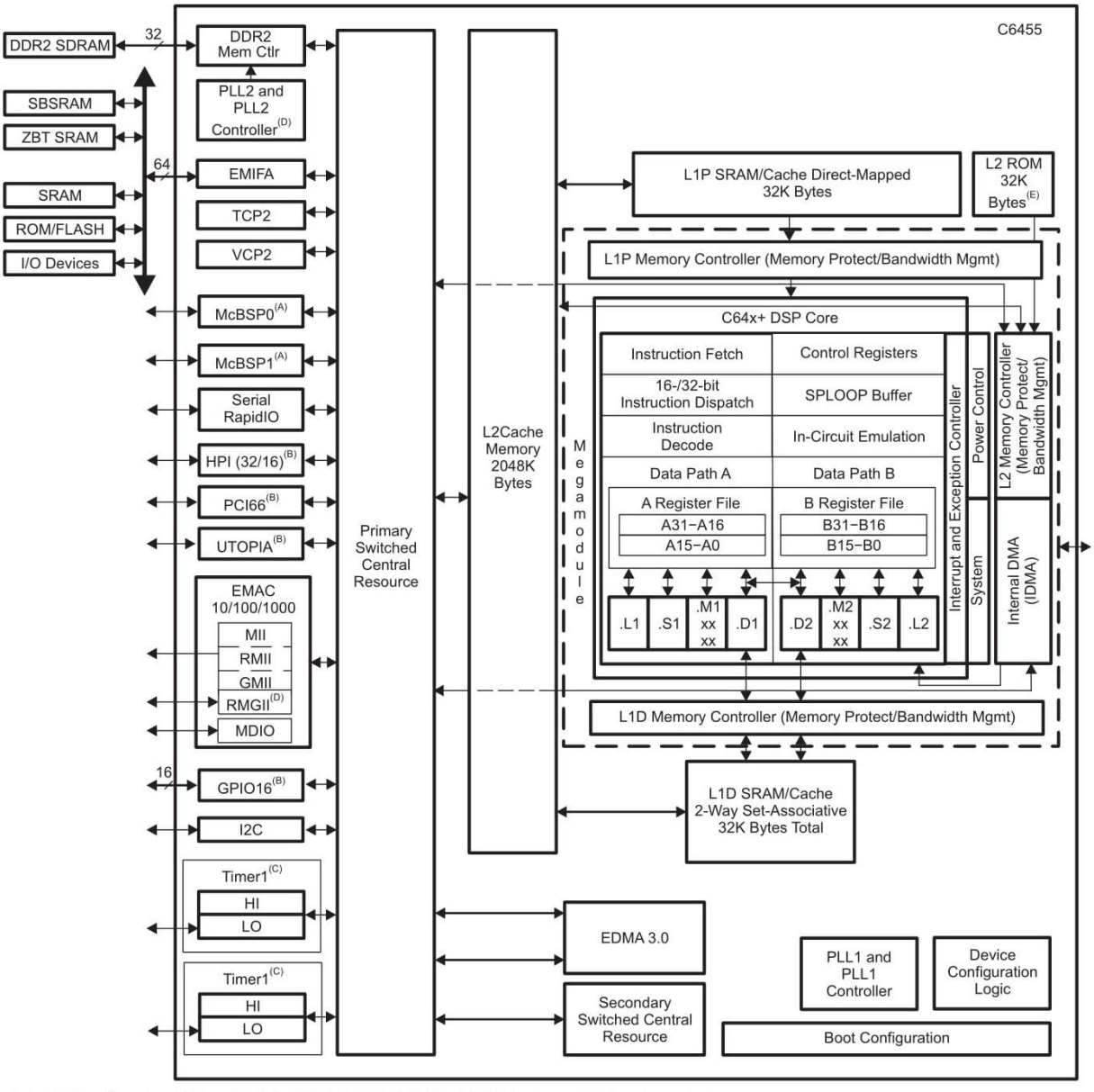


Рис. 1.4. Блок-схема процесору DSP TMS320C6455

2. Ядро ЦСП TMS320C64x

– спеціалізована інструкція SPLOOP;

– компактні інструкції (32- / 16- розрядів);

– розширення набору інструкцій;

– обробка винятків.

3. Архітектура TMS320C64x з мегамодульною пам'яттю L1 / L2

– кеш-пам'ять програм L1P розміром 256 Кбіт (32 Кбайт) (таблична організація);

– кеш-пам'ять даних L1D розміром 256 Кбіт (32 Кбайт) (асоціативний набір);

– об'єднане ОЗУ / кеш-пам'ять L2 розміром 16Мбіт (2048 Кбайт);

– лічильник відміток часу.

4. Розширений співпроцесор дешифратора Viterbi (VCP2)

– підтримка до 694 AMR 7.95 Кбіт / с;

– програмовані параметри коду.

5. Розширений співпроцесор турбодешіфратора (TCP2)

– підтримка до восьми 2 Мбіт / с 3GPP (6 ітерацій);

– програмовані параметри турбокодів і параметри дешифрування.

6. Підтримка прямого (Little Endian) і зворотного (Big Endian) порядку байта.

7. Інтерфейс 64-розрядний / 133 МГц зовнішньої пам'яті

– безпосереднє підключення до асинхронної пам'яті (статичне ОЗУ, флеш-пам'ять і СП ПЗУ) і синхронної пам'яті (SBSRAM і ZBT SRAM);

– підтримка підключення до стандартних синхронних пристроїв і спеціалізованої логіки (FPGA, CPLD, ASIC);

– загальний адресуємий простір зовнішньої пам'яті 32 Мбайт.

8. Контролер пам'яті 32-розрядний DDR2 (DDR2-500 SDRAM).

9. Контролер EDMA (64 незалежних канали).

10. Чотири 1-х послідовних каналу RapidIO (або один 4x), сумісний з версією 1.2

– швидкості передачі 1.25-, 2.5-, 3.125 Гбіт / с;

– передача повідомлень, підтримка DirectIO, розширення для управління обробкою помилок і управління завантаженням каналу;

– введення-виведення, сумісний з IEEE 1149.6.

11. Інтерфейс хост-порту 32/16-разр (HPI).

12. Ведучий / підлеглий інтерфейс PCI 32-разр. / 66 МГц / 3.3В-ий, що відповідає вимогам PCI 2.3.

13. Одна шина I2C.

14. Два багатоканальних буферизованих послідовних порти (McBSP).

15. Контролер 10/100/1000 Мбіт / с Ethernet MAC (EMAC)

– сумісність з IEEE 802.3;

– підтримка декількох медіа-незалежних інтерфейсів (MII, GMII, RMII і RGMII);

– 8 роздільних каналів передачі (TX) і 8 роздільних каналів прийому (RX).

16. Два 64-розрядних таймера загального призначення, які можуть працювати як чотири 32-розрядних таймера.

17. Універсальний інтерфейс тестування і роботи для ATM (UTOPIA)

– підлеглий ATM-контролер 2 рівня UTOPIA;

– 8-розрядний прийом і передача на частоті до 50МГц в одному напрямку;

– формат комірки до 64 байт, що визначається користувачем.

18. Загального призначення 16 ліній введення-виведення.

19. Системна ФАПЧ і контролер ФАПЧ.

20. Додаткова ФАПЧ і контролер ФАПЧ для EMAC і контролера пам'яті DDR2.

21. Граничне сканування відповідно до? IEEE-1149.1 (JTAG).

22. Корпус з матричним розташуванням сферичних виводів (BGA) 697-вивідний (суфікс ZTZ), крок висновків 0.8 мм.

23. КМОП-технологія 0.09 мкм з 7-рівневою мідною металізацією.

24. Живлення 3.3-, 1.8-, 1.5-, 1.2-В введення-виведення, внутрішнє живлення 1.2В.

Ядро процесора. Ядро ЦСП C64x складається з 8 функціональних блоків, двох регістрових файлів і двох каналів даних. За аналогією з іншими ЦСП C6000, два з восьми функціональних блоків є множинними пристроями або блоками ".M". Кожен блок .M у C64x виконує чотири множення-накопичення 16 x 16 за цикл синхронізації. Якщо використовується тактова частота 1 ГГц, то це означає продуктивність 8 млрд. 16-разр. множень-накопичень в секунду. Крім того, кожен множинний пристрій ядра C64x може обчислити одне множення-накопичення в форматі 32 x 32 або чотири множення-накопичення в форматі 8 x 8 за цикл синхронізації.

Послідовний інтерфейс. C6455 містить послідовний інтерфейс RapidIO. Даний високопродуктивний пристрій істотно покращує системні робочі характеристики і знижує вартість системи при роботі з програмними додатками, де потрібна установка на платі декількох ЦСП, наприклад, відео або телекомунікаційне обладнання, медичне обладнання або системи обробки зображень.

Організація пам'яті. ЦСП C6455 інтегрує велику кількість пам'яті, яка організована як дворівнева система. Пам'ять програм і пам'ять даних 1 рівня (L1) має розмір 32 Кбайт. Дана пам'ять може бути налаштована як табличне ОЗУ, кеш-пам'ять або деяка комбінація двох. У режимі кеш-пам'яті пам'ять програм L1 (L1P) є кеш-пам'яттю з табличним доступом (Direct Mapped Cache), а пам'ять даних L1 (L1D) є кеш-пам'яттю з 2-канальним асоціативним доступом. Пам'ять 2 рівня (L2) спільно використовується в якості пам'яті програм і даних із загальним розміром 2 Мбайт. Пам'ять L2 може також функціонувати, як табличне ОЗУ, кеш-пам'ять або деяке сполучення двох. Модуль C64x також містить 32-розрядний конфігураційний порт, контролер внутрішнього ПДП, системні компоненти для управління скиданням / завантаженням, управління перериваннями / винятками, управління зниженням потужності і 32-розрядний таймер.

Склад периферійних пристроїв. До складу периферійних пристроїв входять: шинний модуль I2C; два багатоканальних буферизованих послідовних порту (McBSP); 8-розрядний підлеглий порт UTOPIA; два 64-розрядних таймера загального призначення, які можуть працювати як чотири 32-розрядних таймера; 16- або 32-розрядний інтерфейс хост-порту (HPI16 / HPI32), який конфігурується користувачем; інтерфейс PCI; 16-вивідний порт введення-виведення загального призначення (GPIO) з програмованими режимами генерації переривань / подій; контролер медіа-доступу 10/100/1000 Ethernet (EMAC), який є ефективним інтерфейсом між ядром процесора C6455 і мережею; модуль управління введенням-виведенням даних (MDIO) (є частиною EMAC), який безперервно опитує всі 32 адреси MDIO з метою перебору всіх фізичних пристроїв в системі; інтерфейс зовнішньої пам'яті (64-розр. EMIFA) для безпосереднього підключення до синхронної і асинхронної пам'яті; а також 32-розрядний інтерфейс DDR2 SDRAM.

Порти управління. Порти I2C дозволяють C6455 легко управляти периферійними пристроями та встановлювати зв'язок з хост-процесором. Крім того, стандартний багатоканальний буферізований послідовний порт (McBSP) може використовуватися для зв'язку з периферійними пристроями, що містять послідовний інтерфейс SPI.

Розширені співпроцесори. C6455 містить два високопродуктивних співпроцесора – розширений співпроцесор дешифратора Viterbi (VCP2) і розширений співпроцесор турбодешіфратора (TCP2), які дозволяють істотно прискорити дешифрацію каналу. VCP2

працює на тактовій частоті ЦСП поділеної на 3 та може декодувати до 694 адаптивних багатошвидкісних (AMR) голосових каналів зі швидкістю 7.95 Кбіт / с. TCP2 працює на частоті ЦСП, поділеної на 3, і може декодувати до 50 каналів зі швидкістю 384 Кбіт / с або до 8 турбокодованих каналів зі швидкістю 2 Мбіт / с (допускаючи 6 ітерацій). TCP2 реалізує алгоритм необхідних 3GPP і 3GPP2, а також для повного програмування довжини посилки і турбочасового ущільнювача. Зв'язок між VCP2 / TCP2 і ЦСПС організована за допомогою контролера EDMA.

1.5. Інструменти програмування мікропроцесорних систем

Інструменти програмування МОС, зокрема ЦСП TMS320C6000 мають специфічні для сигнального процесору IT-інструкції, які підтримуються на мові C++. Цей набір інструкцій призначений для 16-розрядних операцій на 32-розрядній архітектурі при використанні Сі-коду. IT забезпечує C6000 сумісність з системами емуляції, які підтримують апаратне та програмне налагодження систем DSP за допомогою кабелю емуляції JTAG.

Файлова модель цифрової обробки. Для програмного моделювання алгоритмів цифрової обробки на процесорах (DSP) TMS320C62x/C67x може бути використана файлова модель, що представлена на рис. 1.5.

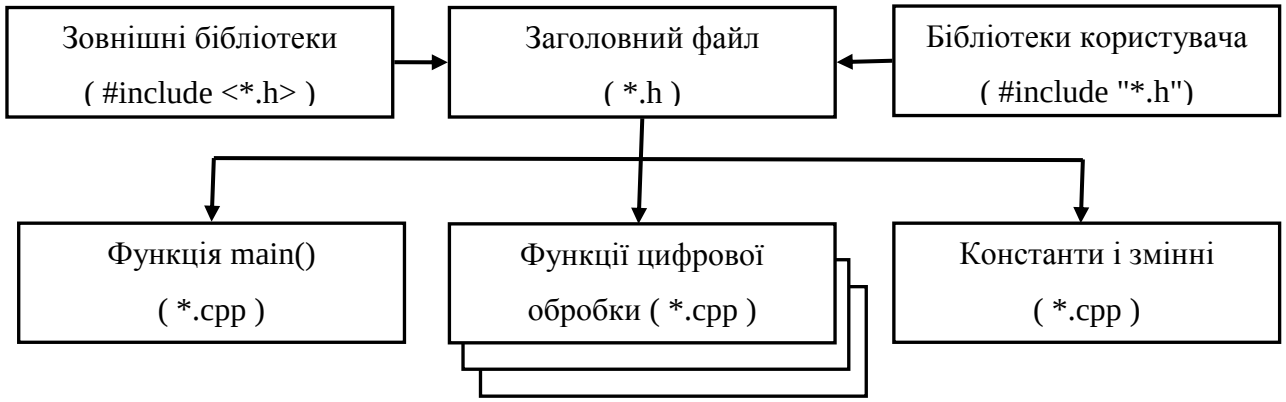


Рис. 1.5. Файлова модель для моделювання алгоритмів цифрової обробки

Суть даного підходу полягає в тому, що вхідні впливи і реакції на них знаходяться в файлах, які називаються тестовими векторами. При розробці конкретного алгоритму (наприклад, цифрової фільтрації) вибирають вплив, реакція на який заздалегідь відома.

Обробляючи вхідний файл за допомогою програмної реалізації алгоритму, отримують вихідний файл і переконуються у відповідності розрахункового та отриманого результатів.

Основні вимоги до файлової моделі прості і полягають в наступному:

1. Наявність тільки одного заголовку (*.h) в програмному коді. В даному файлі проводиться підключення всіх необхідних зовнішніх бібліотек, оголошення констант, масивів, функцій, макросів, призначених для користувача типів і т.д.;

2. Створення контекстної структури алгоритму, що включає всі необхідні для функціонування програми глобальні покажчики, константи, змінні і т.д.;

3. Відсутність статичних і динамічних змінних і констант. Якщо їх наявність необхідна, вони не повинні бути глобальними і мають включаються в контекстну структуру;

4. Створення глобальних змінних, констант, структур і масивів проводиться в одному окремому файлі (*.cpp). Всі глобальні змінні, константи і т.п. повинні бути оголошені в одному заголовному файлі;

5. Кожна спеціальна функція має розташовуватися в окремому файлі (*.cpp) і повинна бути оголошена у заголовку;

6. Обмін даними між спеціальними функціями здійснюється через покажчик на контекстну структуру проекту;

7. Програмний код функцій має ґрунтуватися на базових операторах мови С, яка задовольняє вимогам стандарту eXpresDSP;

8. Функції, що реалізують алгоритм цифрової обробки, не повинні бути прив'язані до конкретної задачі. Обмін даними з периферійними пристроями введення / виведення організовується через вхідний і вихідний буфер;

9. Обробка масивів здійснюється через покажчики на ці масиви, включені в контекстну структуру;

10. Функція main() не повинна включати програмний код, який реалізує алгоритм цифрової обробки сигналів. Завдання даної функції – заповнити вхідний буфер даними з файлу, викликати основну функцію алгоритму цифрової обробки і записати результат з вихідного буфера у вихідний файл.

Засоби проектування. ЦСП TMS320C6455 підтримується повним набором засобів для проектування, в т.ч. новим Сі-компілятором, оптимізатором асемблерування для

спрощення програмування і розподілу процесорного часу, а також Windows-інтерфейсом відладчика Texas Instruments (TI) Code Composer Studio з вихідним кодом програми.

Програмний код, що реалізовано в середовищі програмування Code Composer Studio (файли з розширенням *.h і *.cpp), необхідно перетворити в виконуваний модуль (файл з розширенням *.exe). Операція створення виконуваного модуля забезпечується роботою компілятору і включає два етапи:

1. Трансляція кожного файлу з кодом програми (транслюються тільки файли з розширенням *.cpp, заголовні файли підключаються автоматично за рахунок директиви #include <*.h>) в об'єктні модулі (файли з розширенням *.obj);

2. Компонування або лінковка об'єктних файлів в виконуваний модуль.

Застосування файлової моделі передбачає максимальну переносимість розробленого програмного Сі-коду між різними сімействами цифрових сигнальних процесорів.

1.6. Засоби проектування DSP Test Integration VIs

Для проектування віртуальних приладів з використанням візуальних компонентів передачі даних через порти введення / виводу (I / O) у системі розробки LabVIEW 2010 використовується спеціалізований Toolbox DSP Test Integration VIs, що містить засоби сполучення ЦСП TMS320C6455 з інтерфейсом користувача у LabVIEW (рис. 1.6).

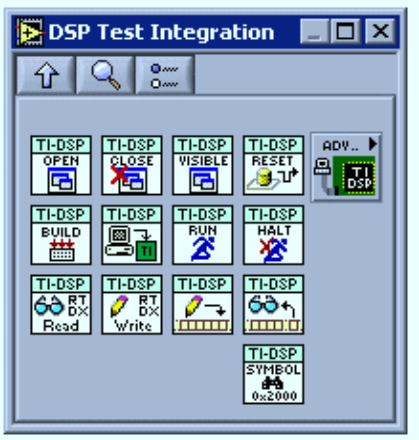


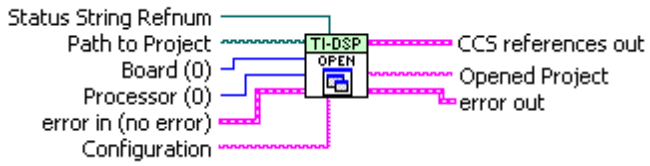
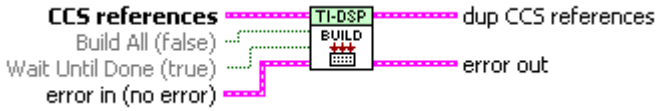
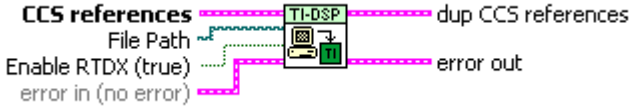
Рис. 1.6. Візуальні компоненти Toolbox DSP Test Integration VIs

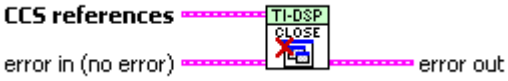
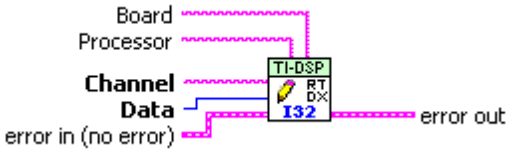
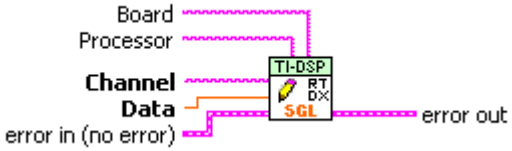
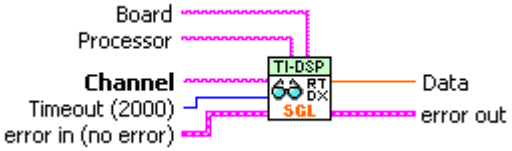
За допомогою компонентів DSP Test Integration VIs реалізується взаємодія віртуального приладу в LabVIEW з програмним забезпеченням у додатку Texas Instruments

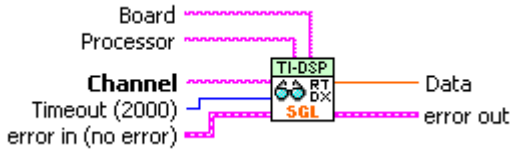
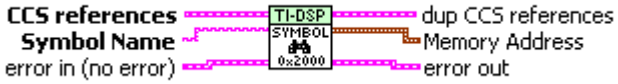
(TI) Code Composer Studio (CCS) через виділені канали RTDX та створюється інтегроване середовище розробки (IDE) для проектування і тестування обладнання.

Перелік компонентів DSP Test Integration VIs в LabVIEW наступний: CCS Open Project, CCS Build, CCS Reset, CCS Download Code, CCS Run, CCS Halt, CCS Close Project, CCS RTDX Write, CCS RTDX Write SGL, CCS RTDX Read, CCS RTDX Read SGL, CCS Symbol to Memory Address. Призначення компонентів DSP Test Integration VIs в LabVIEW представлено у таблиці 1.1.

Табл. 1.1. Призначення компонентів DSP Test Integration VIs

Компоненти DSP Test Integration VIs	*Призначення*
CCS Open Project 	Запуск коду IDE Composer Studio. Якщо CCS вже відкритий, проект (*.pjt) відкривається без запуску іншої копії CCS.
CCS Build 	Запускає операцію збірки проекту в CCS IDE, заново створюючи скомпільований .OUT файл.
CCS Reset 	Зупиняє виконання цільового коду на процесорі TMS320C6455 і відновлює регістри процесора до їх значень за замовчуванням.
CCS Download Code 	Завантажує скомпільований .OUT файл у процесор TMS320C6455, або можна завантажити раніше складений .OUT файл, вказавши шлях до нього.

Компоненти DSP Test Integration VIs	*Призначення*
CCS Run 	Запускає .OUT файл у CCS IDE на процесорі TMS320C6455.
CCS Halt 	Зупиняє .OUT файл, який працює на процесорі TMS320C6455.
CCS Close Project 	Закриває файл відкритого проекту (*.pjt) і закриває всі посилання на CCS IDE.
CCS RTDX Write 	Запис цифрових даних в канал RTDX.
CCS RTDX Write SGL 	Запис цифрових даних в канал RTDX у форматі SGL.
CCS RTDX Read 	Зчитування числових даних з каналу RTDX.

Компоненти DSP Test Integration VIs	*Призначення*
CCS RTDX Read SGL 	Зчитування числових даних з каналу RTDX у форматі SGL.
CCS Symbol to Memory Address 	Повертає адреси пам'яті символу для цільового коду завантаженого на процесорі TMS320C6455.

Інструментарій інтеграції компонентів DSP Test Integration VIs в LabVIEW включає в себе ідентифікатори бібліотечних функцій в CCS та VI, що використовують технологію програмного забезпечення TI Real-Time Data Exchange ™ (RTDX ™) для обміну даними з процесором DSP TMS320C6455 через канали RTDX. Інструментарій для інтеграції DSP з LabVIEW також включає в себе засоби LabVIEW Debugging Workbench для реалізації зв'язку RTDX ™, який можна використовувати для взаємодії з каналами RTDX на платформах розробників, що підтримують RTDX.

1.7. Особливості розподілених систем

Класифікація розподілених систем залежить від реалізованого в них апаратного і програмного забезпечення. Класифікація, в основі якої лежить апаратне забезпечення, залежить від того, яким чином окремі компоненти розподіленої системи з'єднані між собою і як ці компоненти взаємодіють.

Комп'ютери, які складають основу апаратного забезпечення розподіленої системи, поділяють на звичайні (однопроцесорні), паралельні (багатопроцесорні) та суперкомп'ютери. Паралельний комп'ютер має набір процесорів, які здатні спільно працювати під час розв'язання обчислювальних задач. Таке визначення достатньо широке і охоплює як мережі робочих станцій, так і паралельні суперкомп'ютери, що мають сотні або тисячі процесорів.

Суперкомп’ютер представляє собою пристрій, який зводить проблему математичних обчислень до проблеми введення/виведення, тобто найбільш неефективними за вимогами швидкодії стають порівняно повільні пристрої введення/виведення.

Характерною ознакою багатьох паралельних архітектур є реалізація доступу до віддаленої пам’яті (інших процесорів у мережі). Отже, з погляду ефективності обчислень необхідно, щоб доступ до локальних даних був більш швидким і частим, ніж доступ до віддалених даних. Таку властивість паралельній архітектурі надає спеціалізоване програмне забезпечення, що забезпечує локальність доступу (locality), яка поряд з паралелізмом і масштабованістю є основною і необхідною вимогою до розробленого паралельного програмного забезпечення.

Архітектури паралельних комп’ютерів для розподілених систем можуть значно відрізнятися одна від одної. Основними трьома компонентами паралельних комп’ютерів є: процесори, модулі пам’яті, комутаційні мережі. Комутаційна мережа з’єднує усі процесори один з одним, а також іноді з модулями пам’яті. Архітектура процесорів, що застосовуються в паралельних комп’ютерах, схожа з процесорами в однопроцесорних системах, хоча сучасна технологія дозволяє розмістити на мікросхемі декілька процесорів. На одній мікросхемі разом із процесором можуть міститися інші складові паралельної системи, які дають змогу підвищити ефективність паралельних обчислень.

Важливою особливістю, яка відрізняє паралельні комп’ютери, є кількість можливих потоків команд. Прикладом реалізації паралельного комп’ютера є функціональний поділ на послідовні та паралельні інструкції та потоки даних, що зумовлює виокремлення чотирьох класів комп’ютерів:

1. Класична архітектура von Neumann – комп’ютери, що позначаються як SISD (Single Instruction Single Data), де використовується єдиний потік даних та єдина інструкція.

2. Паралельні та розподілені архітектури – комп’ютери, що позначаються як MIMD (Multiple Instruction Multiple Data), де використовуються паралельні потоки даних та множинні інструкції.

3. Змішані форми архітектури, що належать класу до комп’ютерів SIMD (єдина інструкція – паралельний потік даних) або, рідше, до комп’ютерів MISD (множинні інструкції – єдиний потік даних).

Таким чином, відповідно до наведеної класифікації комп'ютерів розрізняють такі архітектури комп'ютерів: SIMD (Single Instruction Multiple Data), MIMD (Multiple Instruction Multiple Data).

Архітектура Single Instruction Multiple – комп'ютер, який має N ідентичних синхронно працюючих процесорів, N потоків даних і один потік команд. У цій архітектурі кожен процесор володіє власною локальною пам'яттю, а мережа, яка з'єднує процесори комп'ютерів, має регулярну топологію (рис. 1.7).

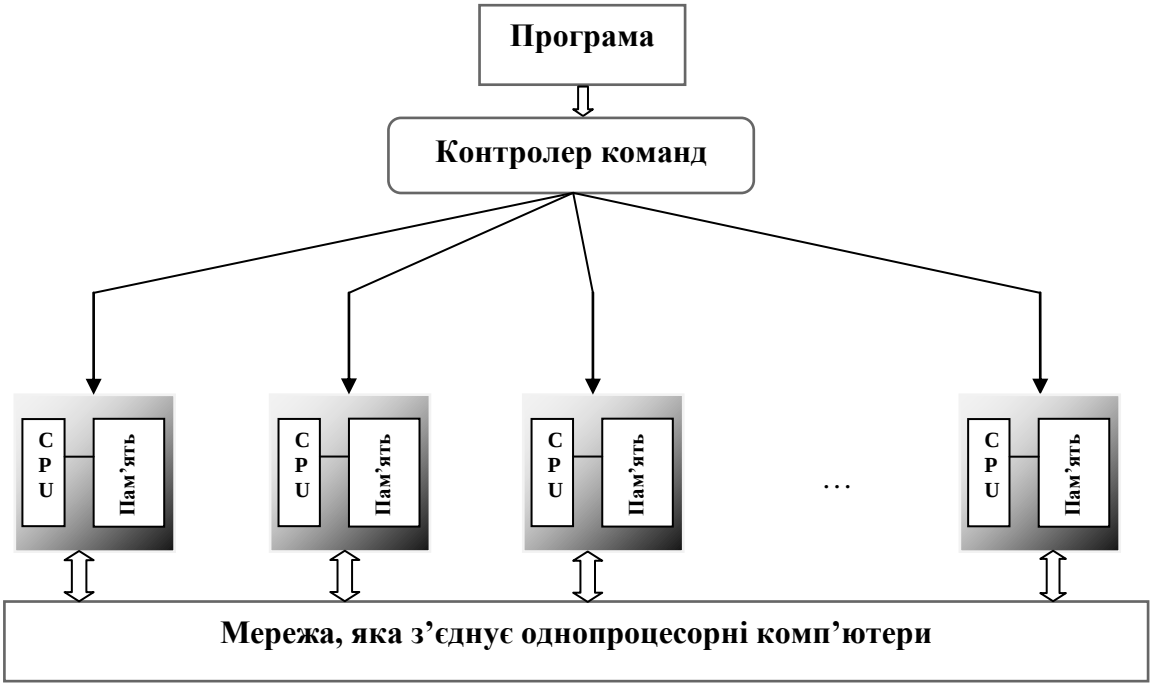


Рис. 1.7. Архітектура SIMD комп'ютера

Процесори в архітектурі SIMD інтерпретують адреси даних як локальні адреси власної пам'яті, або як глобальні адреси, що модифіковані додаванням локальної базової адреси. Усі процесори отримують команди від єдиного центрального контролера команд і працюють синхронно, таким чином, що на кожному кроці всі процесори системи виконують одну команду над даними з локальної пам'яті.

Архітектура Multiple Instruction Multiple – комп'ютер, який має N процесорів, N потоків команд і N потоків даних. У цій архітектурі кожен процесор функціонує під

керуванням власного потоку команд таким чином, що такий комп'ютер може паралельно виконувати різні програми з локальної пам'яті (рис. 1.8).

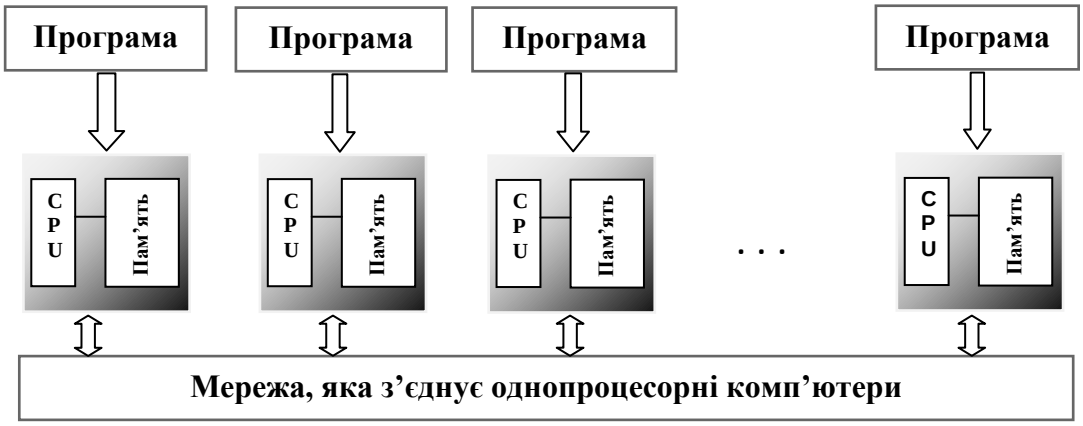


Рис. 1.8. Архітектура MIMD комп'ютера

Архітектури MIMD також класифікують за наступними характеристиками:

– фізичною організацією пам'яті;

– способом доступу до модулів пам'яті (залежно від того, чи має процесор власну локальну пам'ять, або звертається до інших блоків пам'яті);

– використанням комутаційної мережі (залежно від того, чи об'єднує комутаційна мережа всі процесори із загальнодоступною пам'яттю).

Виходячи з методів доступу до пам'яті, організації пам'яті, розрізняють такі типи паралельних архітектур (MIMD):

1. Архітектура з розподіленою пам'яттю (distributed memory), в якій всі процесори об'єднані в мережу, кожен процесор має доступ лише до локальної пам'яті, а доступ до віддаленої пам'яті здійснюється тільки за допомогою системи обміну повідомленнями.

2. Архітектура зі спільною пам'яттю (true shared memory), в якій всі процесори спільно звертаються до загальної пам'яті переважно через єдину шину або ієрархію шин.

3. Архітектура з віртуальною спільною пам'яттю (virtual shared memory), в якій загальної пам'яті немає, кожен процесор має локальну власну пам'ять, але може звертатися до локальної пам'яті інших процесорів, використовуючи глобальну адресу.

В архітектурі з віртуальною спільною пам’яттю, якщо глобальна адреса вказує не на локальну власну пам’ять, то доступ до неї реалізується за допомогою повідомлень, які пересилаються з малою затримкою через вузли мережі, що з’єднує процесори.

Паралельні обчислювальні системи класу MIMD мають три підкласи: симетричні мультипроцесори (SMP), масово-паралельні системи (MPP), кластери (High performance computing clusters, HPC). В основу цієї класифікації паралельних обчислювальних систем покладено структурно-функціональний підхід.

Симетричні мультипроцесори (SMP) складаються із сукупності процесорів, що мають однакові можливості доступу до зовнішніх пристроїв і пам’яті та функціонують під керуванням однієї операційної системи. Всі процесори SMP-систем мають розділювану загальну пам’ять з єдиним адресним простором. Прикладом SMP-системи є однопроцесорні комп’ютери.

Кластерна система (HPC) складаються з модулів, що об’єднані системою зв’язків або розділюваними пристроями зовнішньої пам’яті, зокрема дисковими масивами. Для створення кластерних систем використовуються спеціалізовані технічні засоби (наприклад AWS фірми NCR, Memory Channel фірми DEC), або дискові масиви з високошвидкісними подвійними контролерами (Wide/Fast) і квадро контролерами (PCI SCSI), а також універсальні локальні та глобальні мережі, як наприклад Ethernet, FDDI (Fiber Distributed Data Interface), в тому числі мережі, які працюють із протоколами TCP/IP. Розмір комп’ютерного кластера варіюється від декількох одиниць до декількох десятків модулів.

Масово-паралельні системи (MPP), на відміну від комп’ютерних кластерів, мають спеціалізовані та більш швидкісні канали зв’язку між обчислювальними модулями, а також можливості з масштабування архітектури. Крім того, MPP-системи характеризуються досить високим рівнем інтерфейсу прикладних програм (API), що підтримується розподіленою операційною системою. Однак програмна підтримка оптимізації завантаження процесорів у MPP-системах менш розвинута у порівнянні з кластерами, що пов’язано з різноманітністю виконуваних програм і тим, що немає функціональних зв’язків між програмами.

Відповідно до організації архітектури комп’ютерів розрізняють вільнозв’язані та сильнозв’язні системи. Сильно зв’язні системи можна класифікувати від часу затримки при передачі повідомлення від одного CPU до іншого, в яких реалізується висока швидкість

передачі. У вільноз'єднаній системі спостерігається низька швидкість передачі та можливі триваліші затримки. Прикладом такої системи є два персональні комп'ютери, що з'єднані модемами і телефонною мережею. Сильнозв'язні системи зазвичай мають поділювану пам'ять (shared memory). Такі системи називають мультипроцесорними, оскільки вони використовують алгоритми паралельного розв'язання окремої задачі та можуть розглядатися як мультикомп'ютерна система (МКС). Для обробки декількох незалежних задач, необхідно використовувати мультикомп'ютерну систему з неподілюваною пам'яттю, яка в ідеалі є справжньою розподіленою системою.

Кожна з категорій мультикомп'ютерних систем може на основі використовуваної структури мережі поділятися на switch-базовані та шинно-базовані. Під шинно-базованою структурою розуміють окрему мережу, засоби передачі інформації (мережевий кабель або бездротовий пристрій), що зв'язують всі комп'ютери мережі. Switch-базовані системи не мають у своєму розпорядженні спеціальної мережі, а навпаки підтримують окремі зв'язки між комп'ютерами, тому в кожному вузлі мережі має міститися пристрій для забезпечення процесу маршрутизації.

У мультикомп'ютерній системі, оскільки локальна пам'ять не так завантажена, значно зменшується вихід повідомлень. Для зменшення кількості затримок усі процесори в системі не зв'язуються безпосередньо (Switch), але можуть опосередковано обмінюватися повідомленнями з іншим процесором. Приклади розглянутих топологій наведено на рис. 1.9.

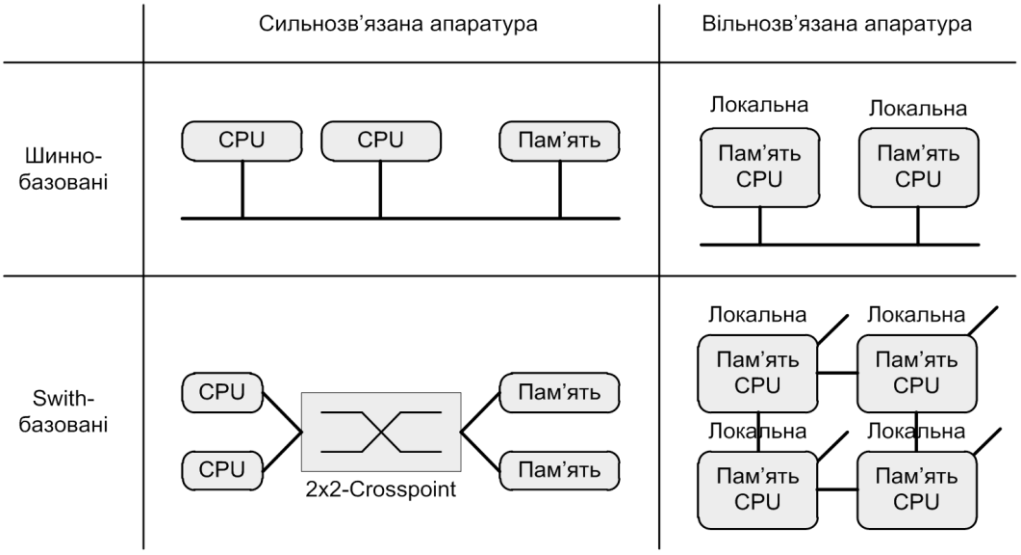


Рис. 1.9. Шинно- і Switch- базовані системи зі спільною пам'яттю і без неї

Таким чином, класифікацію мультикомп'ютерних систем здійснено винятково за ознаками реалізації апаратної частини, а такі системи поділено на чотири групи: вільноз'єднані або сильнозв'язні, шинно- та Switch-базовані.

Аналогічно до класифікації за апаратною складовою також використовують класифікацію мультикомп'ютерних систем за характеристиками програмного забезпечення. Залежно від ступеню зв'язності програмного коду, архітектуру цих систем поділяють на вільнозв'язану та сильнозв'язну.

Застосування вільнозв'язаного програмного забезпечення дозволяє комп'ютерам і користувачам розподіленої системи виконувати незалежну одну від іншої роботу, а також в деяких випадках, якщо буде потреба, сумісно використовувати обладнання. Сильнозв'язне програмне забезпечення дозволяє виконувати одну розроблену програму на різних комп'ютерах одночасно.

Для практичної реалізації розподілених систем спочатку створюється мережева операційна система, потім мультипроцесорна операційна система і розподілена операційна система. Застосування мережевої операційної системи передбачає, що кожний користувач має окрему робочу станцію (Workstation) із власною операційною системою. В цьому разі комунікаційне обладнання використовується для доступу до спільних файлів.

Мультипроцесорна операційна система часто використовують для спеціальних цілей, наприклад для спеціалізованої системи ведення банку даних. Характерною ознакою цієї системи є підтримка окремого процесу для доступу до спільної пам'яті. Комунікація між окремими компонентами цієї системи відбувається для узгодження спільних процесів інформаційного обміну.

Застосування розподіленої операційної системи створює у користувача ілюзію віртуального комп'ютера, у якій вся мережа представляється у вигляді єдиного комп'ютера, де зберігаються вся інформація та прикладні програми. Комунікація в такій системі представляється у якості обміну повідомленнями.

Всі розподілені системи поділяють на наступні категорії:

1. Периферійні системи, які являють собою групу комп'ютерів, що мають спільне функціональне призначення та зв'язані з однією (центральною) машиною. Навантаження, що

припадає на периферійні обчислювальні процеси поділяється із центральним процесором, які переадресовують йому всі повідомлення до операційної системи. Мета використання периферійної системи полягає у підвищенні загальної продуктивності мережі, а також у можливості виділення вільного процесора для одного процесу в операційному середовищі. Периферійна система запускається як окремий модуль, але не має реальної автономії, за винятком процесів розподілення локальної пам'яті та диспетчеризації.

2. Розподілені системи типу «Newcastle», які дозволяють використовувати мережевий зв'язок за іменами дистанційних файлів у бібліотеці. Дистанційні файли мають унікальну специфікацію (складне ім'я), яка являє собою вказівник шляху пошуку і містить спеціальні символи або додатковий компонент імені, що передує кореню файлової системи. Цей метод дозволяє реалізувати мережевий зв'язок без внесення змін у ядро системи, внаслідок чого він є більш простим, ніж інші методи.

3. Абсолютно прозорі розподілені системи, в яких для роботи з файлами, що розташовані на інших машинах, необхідно вказати стандартні складні імена файлів. Маршрути пошуку дистанційних файлів вказано в їх складних іменах, а процес розпізнання цих файлів як дистанційних є функцією ядра.

Відкрита розподілена система (open distributed system) являє собою систему, що має стандартні служби та засоби доступу до ресурсів для широкого кола користувачів, які використовують стандартну семантику і синтаксис всіх протоколів взаємодії. Всі протоколи взаємодії компонентів розподіленої системи ґрунтуються на загальнодоступних стандартах, що дозволяє використовувати для створення програмних компонент різні засоби розробки та операційні системи. Кожна компонента має повну та точну специфікацію своїх сервісів і може бути створена незалежними розробниками програмного забезпечення.

Важливою характеристикою відкритих розподілених систем є наявність загальних специфікацій інтерфейсів користувача, які підтримуються службами та засобами доступу до ресурсів розподілених систем, використовуються для реалізації інтерфейсів різними виробниками програмного забезпечення. Такі специфікації інтерфейсів є однозначними для різних реалізацій розподілених операційних систем.

1.8. Операційні системи та їх розподіленість

Розподілена операційна система дозволяє здійснювати ефективний обмін даними та керування процесами під час взаємодії програмних компонентів, які містяться на одному або на різних комп'ютерах у мережі.

Найважливішими програмними компонентами розподілених систем є спеціалізовані операційні системи та системи проміжного рівня. Основні дані щодо призначення розподілених операційних систем і мережевих операційних систем та засобів проміжного рівня наведені у табл. 2.1.

Таблиця 2.1. Опис розподілених і мережних операційних систем

Система	*Опис*	*Основне призначення*
Розподілені операційні системи	Сильнозв'язні операційні системи для гомогенних мультикомп'ютерних систем і мультипроцесорів.	Керування та приховання апаратним забезпеченням
Мережеві операційні системи	Слабкозв'язні операційні системи для гетерогенних мультикомп'ютерних систем (локальних мереж або глобальних мереж)	Доступ до локальних служб віддаленим клієнтам
Засоби проміжного рівня	Додатковий рівень над мережевою операційною системою, на якому реалізовані служби загального призначення	Забезпечення прозорості ресурсів у мережі

Операційні системи (ОС) для розподіленої системи можна поділити на декілька категорії: слабкозв'язні та сильнозв'язні системи. Слабкозв'язні системи являють собою набір операційних систем, які функціонують спільно і кожна з яких функціонує на локальному комп'ютері, надаючи доступ до власних служб для інших комп'ютерів мережі. Сильнозв'язні системи являють собою операційні системи, які переважно працюють з єдиним глобальним уявленням ресурсів, якими керує ведучий комп'ютер.

Усі сильнозв'язні операційні системи зазвичай є розподіленими операційними системами (Distributed Operating System, DOS), що використовуються для керування мультипроцесорними та гомогенними мультикомп'ютерними системами. Основна мета розподіленої операційної системи полягає у забезпеченні функцій керування апаратним забезпеченням, яке використовує набір обчислювальних процесів.

Слабкозв'язні мережеві операційні системи (Network Operating Systems, NOS) призначені для керування гетерогенними мультикомп'ютерними системами. Ці системи відрізняються від традиційних операційних систем тим, що локальні служби є доступними для віддалених клієнтів, хоча керування апаратним забезпеченням залишається основним завданням мережних операційних систем. Мережеві ОС (NOS) надають в розпорядження віддаленим клієнтам локальні служби, тому такі ОС мають спеціальний сервіс віддаленого доступу – RAS (Remote Access Service).

Виділяють два типи розподілених операційних систем: мультикомп'ютерну операційну систему (multicomputer operating system), яку розроблено для гомогенних мультикомп'ютерів, та мультипроцесорну операційну систему (multiprocessor operating system), яка керує ресурсами мультипроцесора.

Набір функціональностей розподілених операційних систем зазвичай повторює функціональності традиційних операційних систем, що призначені для комп'ютерів з одним процесором, за винятком того, що розподілені операційні системи підтримують роботу декількох процесорів.

Традиційні операційні системи були розроблені для керування комп'ютерами з одним процесором, тому їх називають однопроцесорними. Основним завданням цих систем є організація доступу прикладних програм і користувачів до поділюваних ними пристроїв, таких як процесор, пам'ять, периферійні пристрої та диски. Поділ ресурсів дає можливість використання одного і того ж апаратного забезпечення різними прикладними програмами, які можуть бути ізольовані одна від одної. Для прикладної програми це виглядає таким чином, що доступні ресурси перебувають у повному розпорядженні користувача, при цьому в одній системі може працювати одночасно декілька прикладних програм, кожна з яких має свій набор ресурсів. У такому випадку звичайна операційна система реалізує віртуальну машину (virtual machine), що надає прикладним програмам засоби мультизадачності. При

спільному використанні ресурсів у віртуальній машині прикладні програми відділяються одна від одної, тому неможливі є ситуації, коли одна прикладна програма може змінити дані іншої програми через те, що вона працює з однаковою частиною загальної пам'яті, де зберігаються дані. Прикладні програми можуть використовувати надані засоби лише таким чином, як передбачено операційною системою. Крім того, традиційна операційна система надає первинні операції зв'язку, які можна застосовувати для пересилання повідомлень між прикладними програмами на різних комп'ютерах.

Розподілена операційна система має повністю контролювати розподілення і використання апаратних ресурсів, тому більшість мультипроцесорів підтримують як мінімум два режими роботи: у режимі користувача (user mode) та у режимі ядра (kernel mode). У режимі користувача доступ до пам'яті та регістрів є обмеженим, тобто за межами набору адрес прикладні програми не можуть працювати з пам'яттю, або звертатися безпосередньо до регістрів пристроїв. У режимі ядра виконується весь набір дозволених інструкцій, при цьому в процесі виконання доступні будь-які регістри і вся наявна пам'ять. На час виконання операційної системи процесор комп'ютера перемикається в режим ядра. Щоб перейти з режиму користувача в режим ядра необхідно зробити системний виклик, реалізований через бібліотеку системних функцій операційної системи. Системні виклики здійснюють лише базові служби операційної системи, а обмеження доступу до регістрів і пам'яті зазвичай реалізується апаратно, тому операційна система може повністю їх контролювати.

Існування двох режимів роботи зумовлює таку організацію операційних систем, за якої практично весь їх код виконується в режимі ядра, що в результаті дозволяє створити великі монолітні програми, які працюють у єдиному адресному просторі. Такий підхід дещо ускладнює процес переналаштування операційної системи, оскільки неможливо замінити або адаптувати програмні компоненти без повного перезавантаження системи, а іноді вимагає повної перекомпіляції та нової установки операційної системи. З погляду вимог до проектування прикладних програм, таких як відкритість забезпечення, надійність або легкість обслуговування, монолітні операційні системи є мало ефективними.

Більш зручною є організація операційної системи у вигляді двох частин. Одна частина системи містить набір модулів для керування апаратним забезпеченням, який може виконуватися в режимі користувача. Наприклад, керування загальною пам'яттю полягає у

відстеженні, які блоки пам’яті вільні, а які виділені під процеси, причому робота системи в режимі ядра необхідна під час налаштування регістрів блока керування пам’яттю.

Друга частина системи має мікроядро (microkernel), яке містить винятково код операційної системи, що виконується в режимі ядра. На практиці мікроядро містить лише код для перемикання процесора з одного процесу на другий, встановлення регістрів пристроїв, роботи з блоком перехоплення апаратних переривань і керування пам’яттю. Крім того, у ньому міститься програмний код, який перетворює виклики від відповідних модулів на рівні користувача операційної системи в системні виклики та повертає результати. Такий підхід зумовлює організацію операційної системи, що представлено на рис. 1.10.

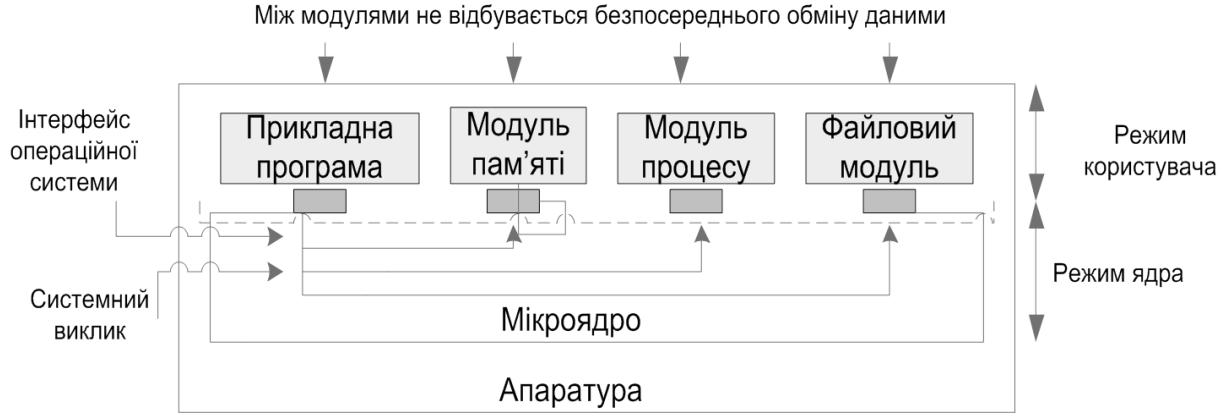


Рис. 1.10. Організація операційної системи з використанням мікроядра

Використання мікроядра надає низку функціональних переваг, найбільш важливими з яких є гнучкість операційної системи, оскільки більша частина коду виконується в режимі користувача, то можна просто замінити один з програмних модулів без повторних компіляції або встановлення системи. Інша перевага використання мікроядра полягає в тому, що модулі рівня користувача можуть розміщуватися на різних комп’ютерах. Цей підхід є зручним для організації роботи однопроцесорних операційних систем на розподілених комп’ютерах.

Подальшим важливим кроком розвитку однопроцесорних операційних систем є організація підтримки декількох процесорів, що мають доступ до використовуваної спільно пам’яті. Такі операційні системи називають мультипроцесорними. Всі структури даних, які необхідні операційній системі для підтримки декількох процесорів та інших пристроїв, розміщуються в пам’яті, що є доступною декільком процесорам. Тому така загальна пам'ять

має бути захищена від паралельного доступу для забезпечення цілісності даних. Проте, у мікроядер є декілька істотних недоліків: по-перше, мікроядро вимагає додаткового обміну даними, що суттєво знижує продуктивність; по-друге, мікроядра працюють інакше, ніж наявні операційні системи.

Багатопроцесорні операційні системи використовуються для підтримання високої продуктивності конфігурацій системи з декількома процесорами. Основне завдання таких систем є забезпечення прозорості та кількості процесорів для прикладних програм. Обмін повідомленнями між різними прикладними програмами або їх частинами, вимагає тих самих примітивів, які застосовують багатозадачні однопроцесорні операційні системи. Всі такі повідомлення працюють із наборами даних у спеціальній області загальної пам'яті, яка використовується спільно, тому забезпечується захист даних від одночасного доступу до них за допомогою примітивів синхронізації, зокрема із використанням двох важливих (еквівалентних) примітивів – семафорів і моніторів.

1.9. Програмне забезпечення проміжного рівня

Розробка розподілених прикладних програм передбачає використання програмного інтерфейсу, який наявний у мережевих операційних системах і є властивим для інтерфейсів локальних файлових систем.

Основне завдання програмного забезпечення проміжного полягає у прихованні відмінностей базових стандартів ОС від прикладного програмного забезпечення, яке використовує низку параметрів або моделей, що характеризують розподілені системи. Проблема такого підходу полягає в тому, що існування розподілу ресурсів є занадто конкретним для реалізації інтерфейсів.

Реалізація програмного забезпечення проміжного рівня дозволяє поєднувати відкритість і масштабованість мережних операційних систем із простотою використання і прозорістю розподілених систем. Таке поєднання властивостей систем дозволяє реалізувати розподілену інформаційну систему із загальною структурою, що наведена на рис. 1.11.

Для забезпечення абстрагування між прикладною програмою користувача та мережевою операційною системою передбачено розміщення додаткового рівня програмної підтримки, який називають проміжним рівнем.

Рис. 1.11. Загальна структура розподілених систем із проміжним рівнем

Система проміжного рівня забезпечує узгоджену роботу ОС і мереж, надаючи можливість використання програмного інтерфейсу системи. Проміжне середовище дає можливість організовувати взаємодію групи комп'ютерів у мережі не порушуючи стек протоколів TCP/IP. Для цього зазвичай використовуються іменовані канали (named pipes) або системні сокети (unix sockets) у POSIX-системах.

Програмне забезпечення проміжного рівня підтримує взаємодію між гетерогенними системами, які на верхньому рівні функціонування системи працюють із прикладними компонентами, а також може отримувати завдання від операційної системи або додаватися до локальної операційної системи. Такий спосіб функціонування дозволяє підтримувати прозорість розподілу. Прикладні програми відокремлюються від деталізації структури функцій операційної системи та ізолюються від взаємодії внутрішніх процесів системи. Розподілені платформи покладені в основу багатьох сучасних конкретних прикладних систем. На цих платформах може одночасно функціонувати декілька прикладних програм.

Основні моделі програмного забезпечення проміжного рівня визначають зв'язок і розподіл в розподілених платформах. Ще однією моделлю організації взаємодії між системами в розподіленому середовищі або прикладними програмами є модель віддаленого виклику процедур RPC (Remote Procedure Call), яка забезпечує виклик служб і процедур з віддалених комп'ютерів. Під час віддаленого виклику процедури RPC параметри передаються на віддалений комп'ютер або сервер, а результат повертається назад. При використанні об'єктного підходу прикладні програми розробляються за допомогою засобів поєднання розподілених об'єктів, причому кожний із цих розподілених об'єктів може реалізувати унікальний інтерфейс, який приховує від кінцевого користувача всі внутрішні деталі реалізації процесу взаємодії. Інтерфейс розподіленого об'єкта передає вихідні дані для реалізації методів. Внутрішній та зовнішній процеси, які взаємодіють між собою, бачать лише свій інтерфейс. Зазвичай розподілені об'єкти розміщуються на одному комп'ютері, а програмне забезпечення проміжного рівня забезпечує дозвіл на доступ до інтерфейсів об'єктів та реалізує цей доступ до них з інших комп'ютерів. Під час виклику методів процесором, інтерфейс об'єктів перетворює дані в повідомлення, яке відсилаються до об'єкту, який виконує програмний метод і повертає результат.

Службою називають прикладну програму, що реалізує кожну із системних функцій на платформі розподілу. Виділяють такі служби платформи розподілу: служби віддаленого доступу (для звертання до розподілених об'єктів і виклику процедур); служби іменування (url); засоби прозорого доступу до віддалених даних (FS, www); засоби зв'язування прикладних програм з базами даних (distributed database); засоби збереження даних або засоби живучості (persistence), наприклад розподілені FS та інтегровані бази даних; засоби розподілених транзакцій, які здійснюють операції записування і зчитування у межах однієї атомарної операції.

При організації розподіленого середовища учасниками взаємодії є окремі сутності, якими можуть виступати прикладні програми, користувачі та інші обчислювальні ресурси. Опис взаємодії двох сутностей представлено у загальній моделі взаємодії клієнт-сервер, у якій одна зі сторін – клієнт ініціює обмін даними, надсилаючи програмні запити другій стороні – серверу, що обробляє запити та у разі потреби надсилає відповідь клієнтові. Модель взаємодії клієнт-сервер представлена на рис. 1.12.

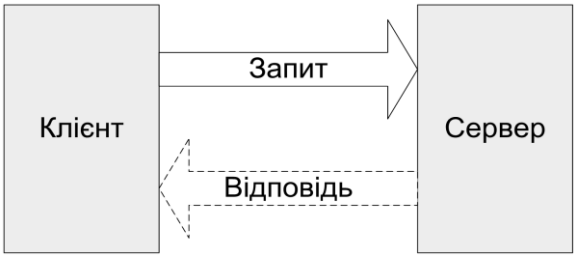


Рис. 1.12. Модель взаємодії клієнт–сервер

У базовій моделі взаємодії клієнт-сервер усі процеси в розподілених системах поділяються на дві групи: процеси, які забезпечують доступ до певної служби за рахунок надсилання запиту та очікування відповіді від сервера, що називають клієнтами (clients); процеси, які реалізують певну службу, наприклад службу бази даних або файлової системи, що називають серверами (servers).

Програмна взаємодія у межах моделі клієнт-сервер може бути як асинхронною, коли клієнт надсилає серверу запит і продовжує виконання коду, не очікуючи відповіді від сервера, або синхронною, коли клієнт очікує на завершення обробки свого запиту сервером. Модель взаємодії сервера і клієнта можна використовувати як основу для опису різних способів взаємодії. У цьому контексті важливою є реалізація взаємодії складових частин програмного забезпечення, які утворюють структуру розподіленої системи.

1.10. Прискорення при паралельних обчисленнях

Поєднання потужних CPU з можливостями організації розподіленого середовища дозволяє вирішувати завдання з великою кількістю послідовних кодів, задовольняючи вимогам паралельних обчислень і обробляючи з високою швидкістю коди, які можуть бути представлена в паралельному вигляді.

Закон Амдала встановлює обмеження на прискорення роботи програми при збільшенні числа обчислювачів. Він виражається наступною формулою:

$$S_n = \frac{1}{\alpha + \frac{1-\alpha}{n}},$$

де α – частка програмного коду, який не може бути представлений в паралельному вигляді, n – кількість обчислювачів, S_p – прискорення роботи програми.

Відповідно до закону Амдала, будь-яка програма має межу прискорення виконання при збільшенні обчислювачів. Виходячи з цього можна прогнозувати, чи буде певний алгоритм давати виграш при паралельному виконанні. Припустимо, наприклад, що частка програми α = 1/3, тобто дві третини операцій в алгоритмі можуть виконуватися паралельно, а третина – не може виконуватися, тоді прискорення становить S_n <3. Таким чином, незалежно від кількості процесорів (ядер) і навіть при ігноруванні всіх витрат на підготовку даних можна прискорити вирішення даного завдання більш, ніж в три рази.

Закон визначає тільки теоретично можливу верхню межу, але на практиці все інакше, оскільки частина ресурсів кожного процесора йде на забезпечення введення і виведення та менеджменту потоків, а шини мають кінцеву пропускну спроможність. При побудові більш потужних систем користувачі прагнуть не скоротити час роботи поточної версії завдання, а перейти до нової версії програмної реалізації, що забезпечує більш високу якість рішення за рахунок збільшення кількості одночасно виконуваних системою потоків обчислень.

Закон Густафсона-Барсіса оцінює максимально допустиме прискорення виконання паралельної програми, в залежності від кількості одночасно виконуваних потоків обчислень і частки послідовних розрахунків. Формула Густафсона-Барсіса виглядає наступним чином:

$$S_n = n + \alpha(1 - n),$$

де α – частка послідовних розрахунків в програмі, n - кількість процесорів.

Відповідно до закону Густафсона зниження загального часу виконання програми поступається обсягу розв'язуваної задачі. Така зміна мети обумовлює перехід від закону Амдала до закону Густафсона. Наприклад, на 100 процесорах програма виконується 20 хвилин. При переході на систему від 1000 процесорами можна досягти часу виконання близько двох хвилин. Однак для отримання більш точного рішення має сенс збільшити обсяг розв'язуваної задачі, тобто при збереженні загального часу виконання користувачі прагнуть отримати більш точний результат. Збільшення обсягу розв'язуваної задачі

призводить до збільшення частки паралельної частини, так як послідовна частина (введення і виведення, менеджмент потоків, точки синхронізації і т.п.) не змінюється.

Принцип заміни простих завдань більш складними, що запропонований Густафсоном, скоріше виключення із загальних правил, ніж повсякденна практика, тому в масових додатках, на які розраховуються багатоядерні процесори, діє закон Амдала. Таким чином, закон прискорення при паралельних обчисленнях визначає верхню межу корисності від збільшення кількості процесорів (ядер) в обчислювальній системі за наступним правилом: якщо задача не може бути представлена в паралельному вигляді через обмеження послідовної частини, то прикладання додаткових зусиль не має ніякого ефекту для розкладу.

Якщо врахувати час, необхідний для передачі даних між вузлами обчислювальної системи, то залежність часу обчислень від числа вузлів матиме максимум (рис. 1.13).

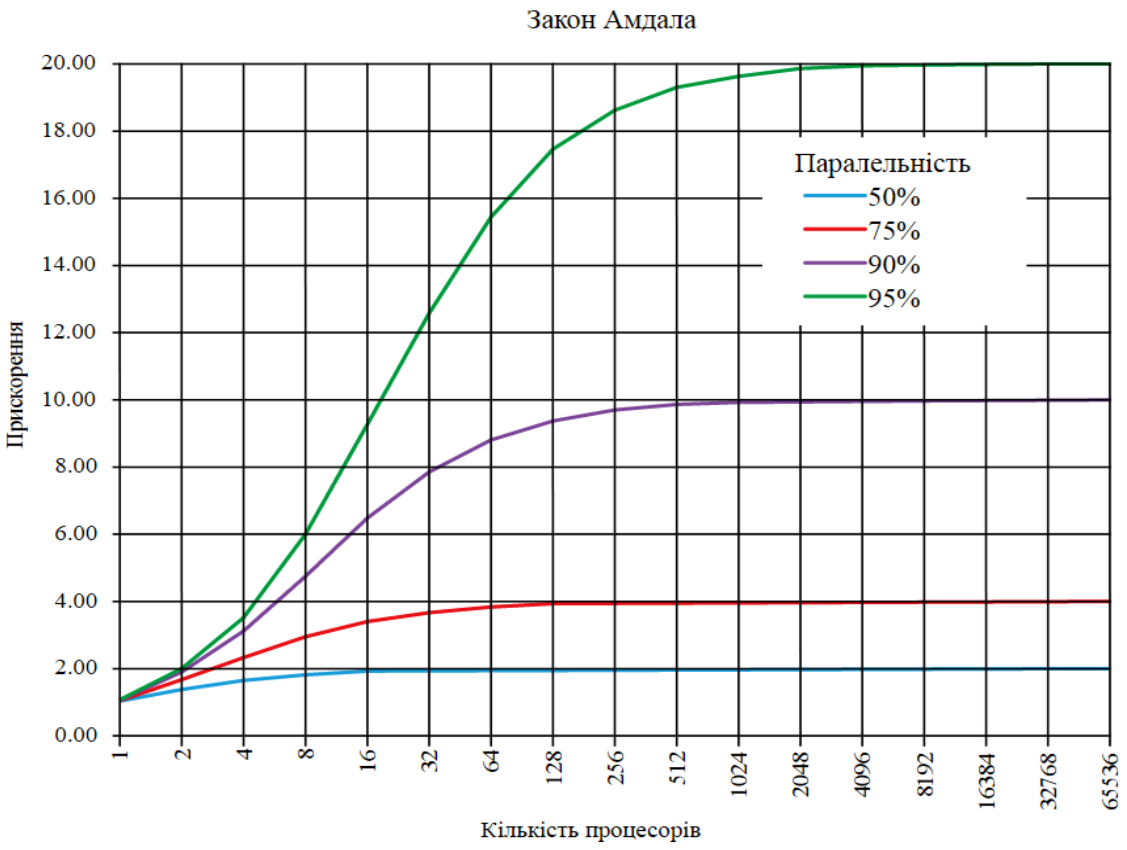


Рис. 1.13 Прискорення при паралельних обчисленнях

Представлені залежності означають, що з певного моменту додавання нових вузлів в розподілену систему буде збільшувати час роботи програми. В першому наближенні обсяг

роботи, яка може бути проведена паралельно, зростає лінійно з ростом числа процесорів в розподіленій системі.

Наприклад, якщо частина програми, що представлена в паралельному вигляді та рівномірно розподіляється по всім процесорам, досягає реалізації в коді 10% послідовних операцій, тоді скільки б процесорів не використовувалося, прискорення роботи програми більш ніж в десять разів ніяк не отримати, а значення 10 – є теоретичною оцінкою самого більшого прискорення, коли ніяких інших негативних факторів немає. Крім того, розпаралелювання веде до певних витрат, яких немає при послідовному виконанні програми. У якості таких витрат можна зазначити додаткові операції, що пов'язані з розподілом програм по процесорам, обміном інформацією між процесорами і т.п.

2. Реалізація алгоритмів паралельних обчислень

Мова програмування C# підтримує паралельне виконання коду через реалізацію багатопоточності методів. Потік – це незалежний шлях виконання частини коду, здатний виконуватися одночасно з іншими потоками.

У деяких аспектах потоки і процеси схожі – наприклад, час поділяється між процесами, що виконуються на одному комп'ютері, так само, як і між потоками одного додатку на мові C#. Ключова відмінність полягає в тому, що процеси повністю ізольовані один від одного, а потоки поділяють загальну пам'ять з іншими потоками цього ж додатка. Завдяки цьому один потік може постачати дані у фоновому режимі, а інший – показувати ці дані в міру їх надходження.

Застосування багато поточності найбільш ефективне в методах, що виконують інтенсивні обчислення. Такі методи можуть виконуватися швидше на багатопроцесорних комп'ютерах, якщо робоче навантаження рознесена по декільком потокам.

Для підтримки багатопоточності необхідно задіяти спеціалізовану бібліотеку:

```
using System;
using System.Threading; // підтримка багатопоточності
```

Програма на C# запускається як єдиний потік, що автоматично створюється операційною системою як "головний" потік:

ThreadTest test = new ThreadTest(); // створюємо загальний об'єкт

Надалі виконання програмного алгоритму стає багатопотоковим за допомогою створення додаткових потоків:

new Thread(test.Go).Start(); // екземплярний метод

У головному потоці створюється новий потік test, який виконує метод Go(), який може потім виконуватися одночасно в головному і в новоствореному потоках:

test.Go();

Для статичних полів працює може використовуватися наступний спосіб розділення даних між потоками:

```
class ThreadTest
{
        static bool done;  //  статичне поле, що розділяється потоками
        static void Main()
        {
                new Thread(Go).Start();  // створюємо загальний об'єкт
                Go();                    // створюється новий потік
        }
        static void Go()                 // екземплярний метод
        {
                if (!done)
                {
                        Console.WriteLine("done");
                        done = true;
                }
        }
}
```

Для забезпечення потокової безпеки необхідно розділити у часі доступ до статичного поля, що використовується декількома потоками. Наприклад, якщо один потік виконує оператор if(), інші потоки можуть виконувати функцію WriteLine, тобто до того як статичне поле done буде встановлено в true.

Один із способів розподілення потоків або синхронізації дій потоків – тимчасове призупинення (блокування) потоків. Очікування ексклюзивного блокування є однією з причин, по яким потік може блокуватися. Отримання ексклюзивною блокування на час читання і запису поділюваних полів на мові C# забезпечується за допомогою оператора lock, що може бити реалізовано наступним чином:

```
class ThreadTest
{
        static bool done;
        static object locker = new object();
        static void Main()
        {
                new Thread(Go).Start();
                Go();
        }
        static void Go()
        {
                lock (locker)  //  блокування потоку на час читання і запису
                {
                        if (!done)
                        {
                                Console.WriteLine("done");
                                done = true;
                        }
                }
        }
}
```

В даному випадку блокування потоку гарантує, що тільки один потік може одночасно виконувати критичну секцію коду, і значення поля "done" буде надруковано тільки один раз. Код, захищений таким чином від невизначеності в плані багатопоточного виконання, називається потокобезпечним.

Інший спосіб розподілення потоків полягає у призупиненні конкуруючих потоків оператором Sleep на заданий проміжок часу:

```
Thread.Sleep(TimeSpan.FromSeconds(30)); // блокування на 30 секунд
```

Також потік може очікувати завершення іншого потоку, викликаючи його метод за допомогою оператора Join:

```
Thread test = new Thread(Go);  // об’єкт Go – статичний метод
test.Start();
test.Join();                   //  очікуємо завершення потоку
```

Після призупинення конкуруючих потоків головний потік продовжує виконання, в той час як робочий потік виконує фонову задачу. З цієї причини в робочому потоці слід запускати задачі, що віднімають більше часу при виконанні завдання.

Таким чином, застосування багатопоточності має сенс, якщо виконуване завдання може зайняти багато часу, так як потрібно очікування відповіді від іншого комп'ютера (сервера додатків, сервера баз даних або клієнта). Запуск такого завдання в окремому робочому потоці означає, що головний потік негайно звільняється для інших завдань.

Проте, надмірне використання багатопоточності забирає ресурси і час CPU на створення потоків і перемикання між потоками. Зокрема, коли використовуються операції читання / запису на диск, більш швидшим може виявитися послідовне виконання завдань в одному або двох потоках, ніж одночасне їх виконання в декількох потоках.

2.1. Методи багатопоточності при обчислення математичних функцій

Ціль роботи – організація потоків на мові C# при паралельному виконанні коду через реалізацію методів багатопоточності для реалізації математичної функції в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

Для створення потоків використовується конструктор класу Thread, що приймає у якості параметра – делегат типу ThreadStart, який вказує метод, що потрібно виконати. Делегат ThreadStart на мові C# визначається наступним чином:

```
public delegate void ThreadStart();
```

Виклик методу Start починає виконання потоку. Потік триває до виходу з виконуваного методу. Наприклад, програмний код для створення делегата ThreadStart має наступний вигляд:

```
class ThreadTest
{
    static void Main()
    {
        Thread t = new Thread(new ThreadStart(Go));
        t.Start();   //  виконати Go() в новому потоці
        Go();       //  одночасно запустити Go() в головному потоці
    }
    static void Go()
    {
        Console.WriteLine("flow method");
    }
}
```

Фоновий потік виконує метод Go() одночасно з головним потоком. Результатом виконування декількох потоків є майже одночасний вивід повідомлення «flow method». Потік, який закінчив виконання, не може бути початий наново, що забезпечує потокову безпеку при паралельному виконанні коду.

Робоче завдання

1. Реалізувати просту математичну функцію на мові C# в середовищі програмування Microsoft Visual Studio 2010 з використанням методів багатопоточності.

2. Дослідити алгоритм виконання програми на мові C# та рівень забезпечення потокової безпеки при паралельному виконанні коду. Описати застосування потокових методів в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та обчислення функції користувача на мові C# необхідно використати конструктор класу Thread та виклик методу Start для виконання потоку. Виконати ініціалізацію даних і змінних у робочому проекті в середовищі програмування Microsoft Visual Studio 2010:

```
// Ініціалізація даних і змінних
using System;
using System.Threading;
namespace Example
{
    class MainClass
    {
        // Ініціалізація масиву значень
        static double[] x = {1, 2, 3, 4, 5};
        static double[] y = {7.1, 27.8, 62.1, 110, 161};
        static int n = 0;
        // Підсумкові коефіцієнти рівняння
        static double a1, b1, a2, b2;
        // Похибка розрахунків
        static double d1, d2;
        public static void Main (string[] args)
        {
            if (x.Length == y.Length) {
                n = x.Length;
```

```
};
// Нормалізація даних
for (int i=0; i<n; i++) {
        x [i] = Math.Log (x [i]);
}
// Створюємо перший потік
Thread thread1 = new Thread(ThreadFunction1);
thread1.Start();
// Створюємо другий потік
Thread thread2 = new Thread(ThreadFunction2);
thread2.Start();
// Підсумкова формула з найменшою похибкою
if (d1 < d2) {
        Console.WriteLine ("Result Point Vector: ");
        Console.WriteLine ("y = " + a1 + "* lnx +"+b1);
} else {
        Console.WriteLine ("Result Point Vector: ");
        Console.WriteLine ("y = " + Math.Pow (Math.E, a2)
        + " * x^ " + b2);
}
}
}
```

2. Реалізувати процедури обчислення функцій ThreadFunction(), що використовуються при запуску потоків thread1 та thread2 паралельно з головним потоком:

```
// Реалізація функцій користувача
static void ThreadFunction1()
{
        // Компоненти для вирішення системи
        double Xi = 0;
        double Xi2 = 0;
        double XiYi = 0;
        double Yi = 0;
```

```
        // Знайдемо необхідні компоненти для вирішення системи
        for (int i=0; i<n; i++) {
                Xi += x [i];
                Xi2 += x [i] * x [i];
                XiYi += x [i] * y [i];
                Yi += y [i];
        }

        // Знайдемо підсумкові коефіцієнти і похибку
        a1 = (Yi * Xi2 * n - XiYi * n * Xi) / (Xi2 * n * n - n * Xi * Xi);
        b1 = (XiYi * n - Yi * Xi) / (Xi2 * n - Xi * Xi);
        d1 = Math.Sqrt (((Yi - a1 * Xi - b1) * (Yi - a1 * Xi - b1))
        / (Yi * Yi));
        Console.WriteLine ("d1 = " + d1);
}
static void ThreadFunction2() {
        double Xi = 0;
        double Xi2 = 0;
        double XiYi = 0;
        double Yi = 0;

        // Нормалізація даних для y = a*x^b
        for (int i=0; i<n; i++) {
                y [i] = Math.Log (y [i]);
        }

        // Знайдемо необхідні компоненти для вирішення системи
        for (int i=0; i<n; i++) {
                Xi += x [i];
                Xi2 += x [i] * x [i];
                XiYi += x [i] * y [i];
                Yi += y [i];
        }
        // Знайдемо підсумкові коефіцієнти і похибку
        a2 = (Yi * Xi2 * n - XiYi * n * Xi) / (Xi2 * n * n - n * Xi * Xi);
        b2 = (XiYi * n - Yi * Xi) / (Xi2 * n - Xi * Xi);
        d2 = Math.Sqrt (((Yi - a2 * Xi - b2) * (Yi - a2 * Xi - b2))
```

```
                / (Yi * Yi));
                Console.WriteLine ("d2 = " + d2);
        }
```

3. За допомогою візуальних компонентів MFC для програмних додатків дослідити алгоритм виконання коду в середовищі програмування Microsoft Visual Studio 2010.

Контрольні питання

1. У чому полягає відмінність між процесами та потоками?
2. Які є способи розподілення потоків або синхронізації дій потоків?
3. Поясніть створення потоків з використанням конструктора класу Thread.
4. Поясніть виклик методу Start для створеного потоку.

2.2. Реалізація потоків для рекурентної функції

Ціль роботи – організація потоків на мові C# при паралельному виконанні коду через реалізацію методів багатопоточності для реалізації рекурентної функції в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

Бібліотека TPL на мові C# має засоби багатопотокового програмування. До їх числа відноситься статичний клас Parallel, який спрощує паралельне виконання коду і надає методи, що дозволяють раціоналізувати обидва види паралелізму – даних і задач. У класі Parallel визначені методи For(), ForEach () і Invoke(), що підтримують паралелізм даних і паралельне виконання двох або більше методів.

Метод Invoke(), що визначений в класі Parallel, дозволяє виконувати один або кілька методів, що вказуються у вигляді його аргументів. Він також масштабує виконуваний код, використовуючи доступні процесори (ядра), якщо є така можливість. Найпростіша форма оголошення цього методу має вигляд:

```
public static void Invoke (params Action [] actions);
```

Методи, що виконуються у якості аргументів Invoke() повинні бути сумісні з описаним раніше делегатом Action, що оголошується наступним чином:

public delegate void Action ();

Метод Invoke() спочатку ініціює виконання, а потім очікує завершення всіх переданих йому методів. Наприклад, застосування методу Parallel.Invoke() для паралельного виконання двох методів має вигляд:

```
using System;
using System.Threading;
using System.Threading.Tasks;
class DemoParallel {
  static void MyMeth1()  //  Метод, виконуваний як задача
  {
        Console.WriteLine("MyMeth1 starting");
        for(int count = 0; count < 5; count++) {
              Thread.Sleep(500);
              Console.WriteLine("In MyMeth1 is counting" + count );
        }
        Console.WriteLine("MyMeth1 stopping");
  }
  static void MyMeth2()  //  Метод, виконуваний як задача
  {
        Console.WriteLine("MyMeth2 starting");
        for(int count = 0; count < 5; count++) {
              Thread.Sleep(500);
              Console.WriteLine("In MyMeth2 is counting" + count );
        }
        Console.WriteLine("MyMeth2 stopping");
  }
  static void Main()  //  Основний потік запущений
  {
```

```
            // Паралельно виконуються два іменованих методи
            Parallel.Invoke(MyMeth1, MyMeth2);
        }       // Основний потік завершено
}
```

В даному прикладі виконання методу Main() припиняється до тих пір, поки не відбудеться повернення з методу Invoke(). Отже, метод Main(), на відміну від методів MyMeth1() і MyMeth2(), не виконується паралельно. Крім того, при застосуванні методу Invoke() відсутня можливість вказати порядок виконання методів від першого і до останнього, і порядок їх виконання може бути не таким, як і в списку аргументів.

Метод For() виконує цикл, що підтримує паралелізм даних. Головна особливість методу For() полягає в тому, що він дозволяє, якщо така можливість є, зробити паралельним виконання коду в циклі. Це, в свою чергу, може привести до підвищення продуктивності.

Найпростіша форма застосування методу For(), наведена нижче:

```
public static ParallelLoopResult For(int fromInclusive, int toExclusive, Action <int> body)
```

Аргумент fromInclusive визначає початкове значення того, що відповідає змінній управління циклом, яка називається також ітераційним, або індексним, значенням; аргумент toExclusive – значення, на одиницю більше кінцевого. На кожному кроці циклу змінна управління циклом збільшується на одиницю. Отже, цикл поступово просувається від початкового значення fromInclusive до кінцевого значенням toExclusive мінус одиниця. Параметр body вказує місце коду, що виконується методом For().

Застосування методу Parallel.For(), що застосовується для організації паралельно виконуваного циклу обробки даних, наведено нижче:

```
using System;
using System.Threading.Tasks;
class DemoParallelFor {
        static int[] data;
        // Метод, що застосовується в якості тіла паралельно виконуваного циклу
        static void MyTrknsform (int i) {
                data[i] = data[i] / 10;
```

```
            if(data[i] < 10000) data[i] = 0;
            if(data[i] > 10000 & data[i] < 20000) data[i] = 100;
            if(data[i] > 20000 & data[i] < 30000) data[i] = 200;
            if(data[i] > 30000) data[i] = 300;
        }
        static void Main () {
            //  Основний потік запущений
            data = new int[100000000];
            //  Ініціалізувати дані в звичайному циклі For()
            for(int i=0; i < data.Length; i++) data[i] = i;
            // Розпаралелити цикл методом For()
            Parallel.For(0, data.Length, MyTransform);
            //  Основний потік завершено
        }
    }
```

На початку цієї програми створюється масив data, що складається з 1000000000 цілих значень. Потім викликається метод For(), до якого в якості "тіла" циклу передається метод MyTransform(). Цей метод складається з ряду операторів, що виконують довільні перетворення в масиві data. Виконання методу For() автоматично розбиває виклики метода MyTransform() на частини для паралельної обробки окремих порцій даних, що зберігаються в масиві. Отже, якщо запустити дану програму на комп'ютері з двома або більше доступними процесорами, то цикл перетворення даних в масиві data може бути виконаний паралельно методом Parallel.For().

Якщо потрібно зупинити паралельний цикл, що виконується методом For(), не звертаючи уваги на будь-які кроки циклу, які ще можуть бути в ньому виконані, то для цієї мети можна скористатися методом Stop().

Робоче завдання

1. Реалізувати просту математичну функцію на мові C# в середовищі програмування Microsoft Visual Studio 2010 з використанням методів багатопоточності.

2. Дослідити алгоритм виконання програми на мові C# та рівень забезпечення

потокової безпеки при паралельному виконанні коду. Описати застосування потокових методів в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та обчислення функції користувача на мові C# необхідно використати конструктор класу Thread та виклик методів Start() для виконання потоку. Виконати ініціалізацію даних і змінних у робочому проекті в середовищі програмування Microsoft Visual Studio 2010:

```
// Ініціалізація даних і змінних
using System;
using System.Threading;
namespace Example
{
        class Thread
        {
                public static double e;
                // Делегат для передачі необхідного методу (g1 та g2) до методу calc()
                public delegate double MyFunc (double x);
                // Точка входу у програму
                public static void Main (string[] args)
                {
                        e =0.01;  //  похибка у розрахунках
                        // Створюємо перший потік
                        Thread thread1 = new Thread(ThreadFunction1);
                        thread1.Start();
                        // Створюємо другий потік
                        Thread thread2 = new Thread(ThreadFunction2);
                        thread2.Start();
```

```
            // Go() – статичний метод
            Thread t = new Thread(Go);
            // Виконати Go() в новому потоці
            t.Start();
            // Одночасно запустити Go() в головному потоці
            Go();
            Thread.Sleep(TimeSpan.FromSeconds(3));  //  блокування на 3 сек.
            t.Join();  //  очікуємо завершення потоку
        }
    }
}
```

2. Реалізувати процедуру обчислення функцій користувача на мові C# у вигляді потоку, що виконує метод Go() одночасно з головним потоком:

```
// Реалізація статичного методу Go(),
static bool done;                    //  статичний тип
static object locker = new object();  //  статичний об’єкт
static void Go()
{
    lock (locker)
    {
        if (!done)
        {
            Console.WriteLine("Done");
            done = true;
        }
    }
}
```

3. Реалізувати на мові C# процедуру обчислення функції calc(), що використовується функціями ThreadFunction1() і ThreadFunction2() при організації потоків:

```
// Реалізація функції f(x) = 2x - cos (x)
```

```
public static double g1 (double x)
{
        double y = 0.5 * Math.Cos(x);
        return y;
}
// Реалізація функції f(x) = x + ln x
public static double g2 (double x)
{
        double y = (2*x - Math.Log(x))/3;
        return y;
}
// Основний метод з розрахунками
public static double calc (double x, MyFunc g)
{
        double temp, d;
        do{
                temp = g(x);
                d = Math.Abs (temp - x);
                x = temp;
                //Console.WriteLine (x);
        }
        while( d>=e );
        return(x);
}
// Організація потоків для функцій
static void ThreadFunction1()
{
        double x = 0.5;
        MyFunc g = g1;
        x = calc (x, g);
```

```
            //Console.WriteLine(x);
            Console.WriteLine("Education 2x - cos (x) = 0");
            Console.WriteLine("Answer: X= {0}, iquel {1}", x,e);
        }
        static void ThreadFunction2(){
            double x = 0.75;
            MyFunc g = g2;
            x = calc (x, g);
            Console.WriteLine("Education x + ln (x) = 0");
            Console.WriteLine("Answer: X= {0}, iquel  {1}", x,e);
        }
```

4. За допомогою візуальних компонентів MFC для програмних додатків дослідити алгоритм виконання коду в середовищі програмування Microsoft Visual Studio 2010.

Контрольні питання

1. У чому полягає відмінність між процесами та потоками?
2. Які є способи розподілення потоків або синхронізації дій потоків?
3. Поясніть створення потоків з використанням конструктора класу Thread?
4. Поясність принципи організації потоків для функцій ThreadFunction().

2.3. Використанням с шаблону для делегованого методу

Ціль роботи – організація потоків на мові C# при паралельному виконанні коду через реалізацію методів багатопоточності при реалізації шаблону для делегованого методу в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

Делегування являє собою групове перетворення методів, що дозволяє привласнити ім'я методу делегату, не вдаючись до оператора new або явного виклику конструктора делегата. Групове перетворення методів можна продемонструвати на наступному прикладі:

```
static void Main() {
        // Сконструювати делегат, використовуючи групове перетворення методів
        StrMod strOp = ReplaceSpaces;  // використати групове перетворення методів
        string str;
        // Викликати методи за допомогою делегата
        str = strOp("This is a simple test.");
        Console.WriteLine("Result string" + str);
        strOp = RemoveSpaces;  // використати групове перетворення методів
        str = strOp("This is a simple test.");
        Console.WriteLine("Result string" + str);
}
```

У наведеному прикладі ім'я методу ReplaceSpaces присвоюється безпосередньо екземпляру делегата strOp, а перетворення методу в тип делегата відбувається автоматично засобами C#. Цей синтаксис може бути поширений на будь-яку ситуацію, в якій метод присвоюється або перетвориться в тип делегата.

Делегати є також гнучкими засобами програмування завдяки двом особливостям: коваріантним і контраваріантним властивостям. Як правило, метод, який передається делегату, повинен мати такий же зворотній тип і сигнатуру, як і делегат. Зокрема, коваріантність дозволяє привласнити делегату метод, зворотним типом якого є клас, похідний від класу, що вказується в повернутому типі делегата. Натомість, властивість контраваріантності дозволяє привласнити делегату метод, типом параметра якого є клас, який є базовим для класу, що зазначений в оголошенні делегата.

Приклад програми, що демонструє властивості коваріантності і контраваріантності, представлений нижче:

```
using System;
class X {
        public int Val;
}
// Клас Y, похідний від класу X
```

```
class Y : X { }
// Цей делегат повертає об'єкт класу X,
// і приймає об'єкт класу Y в якості аргументу
delegate X ChangeIt (Y obj);
class CoContraVariance {
        // Цей метод повертає об'єкт класу X,
        //  і має об'єкт класу X в якості параметра
        static X IncrA (X obj) {
                X temp = new X();
                temp.Val = obj.Val + 1;
                return temp;
        }
        // Цей метод повертає об'єкт класу Y,
        //  і має об'єкт класу Y в якості параметра
        static Y IncrB(Y obj) {
                Y temp = new Y();
                temp.Val = obj.Val + 1;
                return temp;
        }
        static void Main() {
                Y Yob = new Y();
                // В даному випадку параметром методу IncrA є об'єкт класу X,
                // а параметром делегата ChangeIt є об'єкт класу Y.
                // Завдяки контраваріантності наступний рядок коду є допустимим
                Changelt change = IncrA;
                X Xob = change(Yob);
                Console.WriteLine("Xob: " + Xob.Val);
                // В цьому випадку типом методу IncrB служить тип об'єкту класу Y,
                // а повертається типом делегата ChangeIt є об'єкт класу X.
                // Завдяки коваріантності наступний рядок коду є допустимим
```

```
            change = IncrB;
            Yob = (Y) change (Yob);
            Console.WriteLine("Yob: " + Yob.Val);
        }
    }
```

В даному прикладі клас Y є похідним від класу X. Делегат повертає об'єкт класу X і приймає як параметр об'єкт класу Y. Метод IncrA() приймає об'єкт класу X в якості параметра і повертає об'єкт того самого класу. Метод IncrB() приймає в якості параметра об'єкт класу Y і повертає об'єкт того самого класу. Але завдяки коваріантності та контраваріантності будь-який з цих методів може бути переданий делегату ChangeIt.

Робоче завдання

1. Реалізувати просту математичну функцію на мові C# в середовищі програмування Microsoft Visual Studio 2010 з використанням методів багатопоточності.

2. Дослідити алгоритм виконання програми на мові C# та рівень забезпечення потокової безпеки при паралельному виконанні коду. Описати застосування потокових методів в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та обчислення функції користувача на мові C# необхідно використати конструктор класу Thread та виклик методу FormatNumber для виконання потоку. Виконати ініціалізацію даних і змінних у робочому проекті в середовищі програмування Microsoft Visual Studio 2010:

```
// Ініціалізація даних і змінних
using System;
using System.Collections.Generic;
using System.Text;
namespace MulticastDelegate
```

```
{
        class Thread
        {
                delegate void FormatNumber(double number);
        }
}
```

2. Реалізувати процедуру обчислення функції FormatNumber() у вигляді потоку, що використовує екземпляри format при виконанні головного потоку:

```
// Обчислення функції користувача
static void Main(string[] args)
{
        FormatNumber format = FormatNumberAsCurrency;
        format += FormatNumberWithCommas;
        format += FormatNumberWithTwoPlaces;
        format(12345.6789);

}
```

3. Реалізувати на мові C# процедуру обчислення методів-делегатів, що використовують екземпляри format при виконанні головного потоку:

```
static void FormatNumberAsCurrency(double number)
{
   Console.WriteLine("A Currency: {0:C}", number);
}
static void FormatNumberWithCommas(double number)
{
   Console.WriteLine("With Commas: {0:N}", number);
}
static void FormatNumberWithTwoPlaces(double number)
{
   Console.WriteLine("With 3 places: {0:.###}", number);
}
```

4. За допомогою візуальних компонентів MFC для програмних додатків дослідити алгоритм виконання коду в середовищі програмування Microsoft Visual Studio 2010.

Контрольні питання

1. У чому полягає відмінність між процесами та потоками?
2. Які є способи розподілення потоків або синхронізації дій потоків?
3. Поясніть створення потоків з використанням конструктора класу Thread?
4. Поясніть використання екземплярів format при виконанні головного потоку.

2.4. Використання методів-делегатів для організації потоків

Ціль роботи – організація потоків на мові C# при паралельному виконанні коду через реалізацію методів багатопоточності для реалізації методів-делегатів в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

Одним з найбільш примітних властивостей делегата є підтримка групової адресації. Для виконання групової адресації необхідно створити список методів, або ланцюжок викликів для методів, які викликаються автоматично при зверненні до делегату. Для цього достатньо отримати примірник делегата, а потім додати методи в ланцюжок за допомогою оператора + або + =.

Реалізацію підтримки делегатом групової адресації представлено на прикладі заміни пропусків дефісами та видалення пропусків у рядку " This is a simple test ":

```
using System;
// Оголосити тип делегата
delegate void StrMod (ref string str);
class MultiCastDemo {
        // Метод замінити пропусків дефісами
        static void ReplaceSpaces(ref string s) {
                Console.WriteLine("Заміна пропусків дефісами");
                s = s.Replace(' ', '-');
        }
```

```
        // Метод видалення пропусків
        static void RemoveSpaces(ref string s) {
                string temp = "";
                int i;
                Console.WriteLine("Видалення пропусків");
                for(i=0; i < s.Length; i++)
                if(s[i] != ' ') temp += s[i];
                s = temp;
        }
        //  Основний потік запущений
        static void Main() {
                // Сконструювати делегати
                StrMod strOp;
                StrMod replaceSp = ReplaceSpaces;
                StrMod removeSp = RemoveSpaces;
                string str = " This is a simple test ";
                // Організувати групову адресацію
                strOp = replaceSp;
                // Звернутися до делегату з груповою адресацією
                strOp(ref str);
                Console.WriteLine("Result string" + str);
                // Видалити метод заміни пропусків і додати метод видалення пробілів
                strOp -= replaceSp;
                strOp += removeSp;
                str = " This is a simple test "; // відновити вихідну рядок
                // Звернутися до делегату з груповою адресацією
                strOp (ref str);
                Console.WriteLine("Result string" + str);
        }  //  Основний потік зупинений
}
```

У цьому прикладі в методі Main() створюються три примірники делегата. Перший з них, strOp, є порожнім, а два інших посилаються на конкретні методи зміни рядків. Потім організовується групова адресація для виклику метода RemoveSpaces(). Ланцюжки викликів є досить ефективним механізмом, оскільки вони дозволяють визначити ряд методів, які виконуються єдиним блоком. Завдяки цьому поліпшується структура деяких видів коду. Крім того, ланцюжки викликів мають особливе значення для обробки подій.

Робоче завдання

1. Реалізувати просту математичну функцію на мові C# в середовищі програмування Microsoft Visual Studio 2010 з використанням методів багатопоточності.

2. Дослідити алгоритм виконання програми на мові C# та рівень забезпечення потокової безпеки при паралельному виконанні коду. Описати застосування потокових методів в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та обчислення функції користувача на мові C# необхідно використати конструктор класу Thread та виклик методу MathOp для виконання потоку. Виконати ініціалізацію даних і змінних у робочому проекті в середовищі програмування Microsoft Visual Studio 2010:

```
// Ініціалізація даних і змінних
using System;
using System.Collections.Generic;
using System.Text;
namespace Delegates
{
   class Thread
   {
```

```
        delegate T MathOp<T>(T a, T b);
    }
}
```

2. Реалізувати процедуру обчислення функції користувача на мові C# у вигляді потоку, що виконує методи-делегати MathOp одночасно з головним потоком:

```
// Обчислення функції користувача
static void Main(string[] args)
{
        double a = 1.0;
        double b = 3.0;
        double result = 0.0;
        // Організація паралельних потоків
        FormatNumber format = FormatNumberAsCurrency;
        format += FormatNumberWithCommas;
        format += FormatNumberWithTwoPlaces;
        // Реалізація методів-делегатів
        List<MathOp<double>> opsList = new List<MathOp<double>>();
        opsList.Add(Add);
        opsList.Add(Divide);
        opsList.Add(Multiply);
        // Виконання функції користувача
        format(12345.6789);
        foreach (MathOp<double> op in opsList)
        {
                result = op(a, b);
                Console.WriteLine("result = {0:N}", result);
        }
}
```

3. Реалізувати на мові C# процедуру обчислення методів-делегатів, що входять до списку opsList у вигляді статичних методів:

```
static double Add(double a, double b)
{
        return a + b;
}
static double Divide(double a, double b)
{
        return a / b;
}
static double Multiply(double a, double b)
{
        return a * b;
}
static void FormatNumberAsCurrency(double number)
{
        Console.WriteLine("A Currency: {0:C}", number);
}
static void FormatNumberWithCommas(double number)
{
        Console.WriteLine("With Commas: {0:N}", number);
}
static void FormatNumberWithTwoPlaces(double number)
{
        Console.WriteLine("With 3 places: {0:.###}", number);
}
```

4. За допомогою візуальних компонентів MFC для програмних додатків дослідити алгоритм виконання коду в середовищі програмування Microsoft Visual Studio 2010.

Контрольні питання

1. У чому полягає відмінність між процесами та потоками?
2. Які є способи розподілення потоків або синхронізації дій потоків?

3. Поясніть створення потоків з використанням конструктора класу Thread?
4. Поясніть послідовність виконання методів-делегатів MathOp.

3. Реалізація технологій "клієнт-сервер"

У базовій моделі клієнт–сервер усі процеси в розподілених системах поділяють на дві групи. Процеси, які прямо реалізують деяку службу, наприклад службу файлової системи або бази даних, є серверами (servers), а процеси, які вимагають служби у серверів через надсилання запиту і подальшого очікування відповіді від сервера, – клієнтами (clients). Взаємодію між клієнтом та сервером називають режимом запит-відповідь (request-reply behavior).

Модель клієнт-сервер була приводом для багатьох дебатів і суперечок, у яких одне з основних питань полягало в тому, як розподілити частини програмного забезпечення між клієнтом і сервером. Зрозуміло, що зазвичай чіткого розуміння немає. Наприклад, сервер розподіленої бази даних може постійно бути клієнтом, який передає запити на різноманітні файлові сервери, відповідальні за реалізацію таблиць цієї бази даних. У такому разі сервер баз даних сам не робить нічого, крім обробки запитів.

Проте, розглядаючи велику кількість прикладних програм типу клієнт-сервер, передбачених для організації доступу користувачів до баз даних, часто рекомендують поділяти їх на три рівні:

– рівень інтерфейсу користувача;

– рівень обробки;

– рівень даних.

Рівень користувацького інтерфейсу містить усе необхідне для безпосереднього спілкування з користувачем, наприклад для керування дисплеєм. До рівня обробки належить прикладне програмне забезпечення, а до рівня даних – дані, з якими доводиться працювати.

3.1. Обчислення простої функції на сервері

Ціль роботи – розробка інтерфейсу користувача і обчислення математичної функції на сервері через реалізацію методів багатопоточності на основі базової моделі клієнт-сервер на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

Програмний додаток для розподіленого завдання складається з двох частин – сервера і клієнта. Клієнт працює тільки з одним сервером, тоді як сервер обслуговує довільне число клієнтів. Серверний програмний додаток підтримує для кожного клієнта стан сесії.

Крім того, необхідно, виконання наступних умов:

– клієнти повинні вибирати завдання для виконання;

– клієнти повинні повідомляти про те, що виконання завдання завершено;

– сервер в реальному часі повинен повідомляти клієнтам про надходження завдань;

– сервер повинен відстежувати завдання, що взяті на обробку і виконання клієнтом.

Застосування засобів .NET Remoting передбачає додавання в програмний додаток, наприклад JobServer, наступних елементів підтримки, що перераховані нижче:

1. реалізація дистанційного типу;
2. вибір серверного домена додатку;
3. вибір моделі активізації;
4. вибір каналу і порту;
5. спосіб отримання метаданих сервера клієнтами;
6. настройка параметрів .NET Remoting для сервера.

Найпростіший в написанні сервер .NET Remoting – це вже існуючий сервер IIS, що використовує служби Windows. Для забезпечення серіалізації об'єкта в .NET Framework необхідно використати атрибут [serializable]. Крім того, створений у додатку віддалений об'єкт JobServerlmpI необхідно зробити базовим на основі класу System.MarshalByRefObject.

Сервер NET Remoting надає два стандартних канали – HttpChannel та TcpChannel. Вибір канального транспорту у багатьох випадках тип транспорту не має значення. При виборі типу каналу необхідно врахувати кілька факторів:

– необхідність пересилати дані через брандмауер;

– проблеми захисту, що викликані пересиланням даних відкритим текстом;

– необхідність засобів захисту, що підтримуються IIS.

Потік повідомлень між клієнтом і сервером при виборі каналу HttpChannel використовує за замовчуванням SOAPFormattef, щоб формат повідомлень став читабельним. Наступний фрагмент коду показує, як налаштовується канал:

```
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
HttpChannel oJobChannel = new HttpChannel (4000);
ChannelServices.RegisterChannel (oJobChannel);
```

Для випадків, коли потрібна максимальна ефективність, інфраструктура серверу .NET Remoting надає транспорт даних на основі сокетів, що використовуються для передачі потоків за протоколом TCP. Тип каналу передачі даних TcpChannel визначено в просторі імен System.Runtime.Remoting.Channeis.Tcp та реалізовує інтерфейси передачі IChannel, IChannelReceiver і IChannelSender.

У наступному фрагменті коду домен програми налаштовується на використання каналу TcpChannel, що очікує надходження вхідних викликів на порту 2000:

```
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
TcpChannel cJobChannel = new TcpChannel (2000);
ChannelServices.Register (cJobChannel);
```

Серверний процес .NET Remoting підтримує два режими серверної активізації: Sigleton і SingleCall. В режимі Singleton в окремий момент часу може бути активний не більш ніж один екземпляр типу. В режимі серверної активізації SingleCall забезпечується підтримка моделі програмування без збереження стану каналу. Для типу, що оголошений як SingleCall, інфраструктура .NET Remoting буде активізувати новий екземпляр.

Наступний фрагмент коду демонструє спосіб конфігурації об'єктного типу в режимі Sigleton в програмному додатку, який реалізує цей об'єктний тип:

```
RemotingConfiguration.RegisterWellKnownServiceType(typeof( SomeMBRType ),
```

```
"SomeURI",
WellKnownObjectMode.Singleton );
```

Зміст функції Main() у програмному додатку JobServer для прикладу реєстрації каналу HttpChannel на порту 4000 має вигляд:

```
static void Main(string[] args)
{
    // реєстрація каналу
    HttpChannel oJobChannel = new HttpChannel( 4000 );
    ChannelServices.RegisterChannel( oJobChannel );
    // реєстрація загальновідомого типу
    RemotingConfiguration.RegisterWellKnownServiceType(typeof( JobServerlmpl ),
    "JobURI",
    WellKnownObjectMode. Singleton );
}
```

У наведеному прикладі спочатку створюється екземпляр класу HttpChannel, його конструктору передається значення 4000, що являє собою номер порту, на якому сервер очікуватиме виклики від клієнтів. Для того щоб канал почав приймати вхідні повідомлення він реєструється з використанням статичного методу ChannelServices.RegisterChannel. Далі виконується конфігурація об'єктного типу в режимі Sigleton для серверної активізації процесу на основі сокетів.

Робоче завдання

1. Реалізувати інтерфейс користувача і обчислення заданої функції на сервері з використанням методів багатопоточності на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

2. Дослідити алгоритм виконання програми на мові C# або C++ та рівень забезпечення потокової безпеки при реалізації методів багатопоточності. Описати застосування базової моделі клієнт-сервер в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та обчислення функції користувача на мові C# або C++ необхідно використати конструктор мережевого класу TcpClientChannel та методи конфігурації каналів передачі даних RegisterChannel і RemotingConfiguration для реалізації методів багатопоточності. У серверній частині програмного проекту реалізувати методи GetTextBoxText() і MultMatrix(), до яких надається доступ клієнтам через межі домену програмних додатків, що підтримують процес віддаленого керування, наприклад:

```
partial class Form_SimpleServer
{
    // region Windows Form Designer generated code
    public string MultMatrix()
    {
        int result = 1;
        int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 1; j++)
            {
                result *= array2D[i, j];
            }
        }
        return result.ToString();
    }
}
```

2. У серверній частині програмного проекту створити інтерфейс для доступу до методів обслуговування IMyServiceInterface і клас обслуговування MyService, який успадковує MarshallByRefObject та забезпечує доступ з клієнтів до методів MultMatrix() і GetTextBoxText() на сервері через межі домену програмних додатків, що підтримують процес віддаленого керування, наприклад:

```
namespace SimpleServer
{
    // Interface to access service methods
    public interface IMyServiceInterface
    {
        string GetTextBoxText();
        string MultMatrix();
    }
    // MyService: Service class that inherits MarshallByRefObject
// Enables access to objects across application domain boundaries that support remoting
    public class MyService : MarshalByRefObject, IMyServiceInterface
    {
        // Exposed interface method to retrieve contents
        public string GetTextBoxText()
        {
            return Program.FormSimpleServer.GetTextBoxText();
        }
        public string MultMatrix()
        {
            return Program.FormSimpleServer.MultMatrix();
        }
    }
}
```

3. У клієнтській частині програмного проекту в класі Form_SimpleClient налаштувати канал клієнта, отримати проксі для віддаленого об'єкта та отримати об’єктне посилання на сервісний інтерфейс, наприклад:

```
public Form_SimpleClient()
{
    InitializeComponent();
    // Set up a client channel.
```

```
        TcpClientChannel clientChannel = new TcpClientChannel();
        ChannelServices.RegisterChannel(clientChannel, false);
        // Obtain a proxy for a remote object
        // Doing it this way requires a recompile if changing port, etc.
        RemotingConfiguration.RegisterWellKnownClientType(
        typeof(IMyServiceInterface), "tcp://localhost:33000/MyServiceUri");
        // Obtain object reference to service interface
        remoteObject = (IMyServiceInterface)Activator
        .GetObject(typeof(IMyServiceInterface), "tcp://localhost:33000/MyServiceUri");
    }
```

4. У клієнтській частині програмного проекту в класі Form_SimpleClient створити сервісний інтерфейс iService та виконати виклик методів MultMatrix() і GetTextBoxText(), що реалізовані на серверній частині програмного проекту, наприклад:

```
    public partial class Form_SimpleClient1 : Form
    {
        // Service interface
        SimpleServer.IMyServiceInterface iService;
        public Form_SimpleClient1()
        {
            InitializeComponent();
            // Create interface object
            iService = new SimpleServer.MyService();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Access method (get text from running service form)...
            textBox1.AppendText(iService.GetTextBoxText() + "\n");
        }
        private void button2_Click(object sender, EventArgs e)
        {
```

```
            textBox1.AppendText(iService.MultMatrix() + "\n");
        }
    }
```

5. За допомогою візуальних компонентів MFC для програмних додатків дослідити алгоритм на основі моделі клієнт-сервер, що реалізований в середовищі програмування Microsoft Visual Studio 2010.

Контрольні питання

1. Які є способи розподілення та синхронізації процесів у мережі?
2. У чому полягає особливість процесів, що створені на основі моделі клієнт-сервер?
3. Поясніть використання конструктора мережевого класу TcpClientChannel.
4. Поясніть процедуру доступу до програмних методів MultMatrix() і GetTextBoxText().

3.2. Метод переходу за заданим вказівником ресурсів

Ціль роботи – розробка інтерфейсу користувача і методу переходу за заданим вказівником ресурсів через реалізацію методів багатопоточності на основі базової моделі клієнт-сервер на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

Об’єкти, що отримують зворотний виклик з домена додатку віддаленого об'єкта, називаються спонсорами. Для реєстрування посилання на віддалені об'єкти, які слід спонсорувати, використовується клас ClientSponsor. Для створення типу, здатного виступати в якості спонсора, необхідно реалізувати інтерфейс ISponsor. Коли посилання на віддалений об'єкт передається методу ClientSponsor.Register, цей метод реєструє екземпляр ClientSponsor в якості спонсора ліцензії віддаленого об'єкта і запам'ятовує посилання на ліцензію віддаленого об'єкта у внутрішній хеш-таблиці. **Т**ип спонсора повинен бути похідним від екземпляру Systern.MarshalByRefObject, який взаємодіє з об'єктами у віддалених доменах додатків за допомогою проксі.

Приклад використання класу ClientSponsor, що посилається на екземпляр типу похідного від MarshalByRefObject представлений нижче:

```
// Використання класу ClientSponsor передбачає,
// що someMBR посилається на екземпляр типу, похідний від MarshalByRefObject
ClientSponsor cp = new ClientSponsor (TimeSpan.FromMinutes (5));
cp.Register (someMBR);
```

Для створення проксі потрібно створити клас, похідний від RealProxy, додавши новий конструктор і закритий об'єкт MarshalByRefObject, як показано нижче:

```
public class HyProxy : RealProxy
{
    MarshalByRefObject _target;
    public MyProxy(Type type, MarshalByRefObject target)
    {
        _target = target;
    }
}
```

При посиланні на реальний об’єкт MarshalByRefObject проксі передає виклики до реального віддаленого об'єкту. Конструктор RealProxy можна викликати і передавати йому тип віддаленого об'єкта, щоб інфраструктура .NET Rernoting могла згенерувати екземпляр TransparentProxy для цього об'єкта.

Для підключення спеціалізованих проксі необхідно використовувати клас ProxyAttribute, який дозволяється застосовувати тільки до об'єктів, що є похідними від ContextBoundObject. Оскільки клас ContextBoundObject є похідним від MarshalByRefObject, спочатку слід створити новий клас, похідний від ProxyAttribute, і перевизначити метод CreateInstance для базового класу:

```
public class MyProxyAttribute : ProxyAttribute
{
    public override MarshalByRefObject CreateInstance (Type serverType)
    {
        MarshalByRefObject target = base.CreateInstance (serverType);
        MyProxy myProxy = new MyProxy(serverType, target);
```

```
            return (HarshalByRefObject)myProxy.GetTransparentProxy();
        }
    }
```

При створенні екземпляра класу MyRemoteObject викликається перевизначення методу CreateInstance, в якому створюється спеціалізований проксі. Потім створюється новий проксі та перетворюється його тип TransparentPrоxy до MarshalByRefObject і повертається результат. Даний прийом дозволяє клієнту викликати new і метод Activator.Createinstance, як зазвичай і при цьому використовувати спеціалізований проксі.

Робоче завдання

1. Реалізувати інтерфейс користувача і обчислення заданої функції на сервері з використанням методів багатопоточності на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

2. Дослідити алгоритм виконання програми на мові C# або C++ та рівень забезпечення потокової безпеки при реалізації методів багатопоточності. Описати застосування базової моделі клієнт-сервер в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та реалізації проекту WebBrowser на мові C# або C++ необхідно використати конструктор мережевого класу TcpClientChannel для конфігурації каналів. У проекті WebBrowser реалізувати метод webBrowser_Navigated( ), який використовує стандартні налаштування програми Internet Explorer для Web-посилання на сервер LocalHost, або переходу на Web-сторінку по заданому URL-адресу, наприклад:

```
namespace Browser
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
        }
        private void webBrowser1_Navigated(object sender,
        WebBrowserNavigatedEventArgs e)
        {
            textBox1.Text = webBrowser1.Url.ToString();
        }
    }
}
```

2. У проекті WebBrowser реалізувати метод webBrowser_DocumentCompleted(), який повинен виконувати перевірку стану true / false отримання Web-сторінки та змінити метод button_Click() для виконання переходу на задану Web-сторінку "http://www.google.com" за замовчанням, наприклад:

```
namespace Browser
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void webBrowser1_Navigating(object sender,
        WebBrowserNavigatingEventArgs e)
        {
            buttonStop.Enabled = true;
        }
        private void webBrowser1_DocumentCompleted(object sender,
        WebBrowserDocumentCompletedEventArgs e)
        {
```

```
            buttonStop.Enabled = false;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = "http://www.google.com";
            webBrowser1.Navigate(textBox1.Text);
        }
    }
}
```

3. За допомогою візуальних компонентів MFC для програмних додатків дослідити алгоритм на основі моделі клієнт-сервер, що реалізований в середовищі програмування Microsoft Visual Studio 2010.

Контрольні питання

1. Які є способи розподілення та синхронізації процесів у мережі?
2. У чому полягає особливість процесів, що створені на основі моделі клієнт-сервер?
3. Поясніть використання конструктора мережевого класу TcpClientChannel.
4. Пояснити використання стандартних налаштувань програми Internet Explorer.

3.3. Авторизація користувача на Web-сервері

Ціль роботи – розробка інтерфейсу користувача і методу авторизації користувача на Web-сервері через реалізацію методів багатопоточності на основі базової моделі клієнт-сервер на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

Об'єктно-орієнтована архітектура .NET Remoting є основою для .NET Web-сервісів. Крім того, .NET Remoting підтримує відкриті Інтернет - стандарти, такі, як Web-сервіси та протокол SOAP (Simple Object Access Protocol). Концепція використання .NET Remoting дозволяє створювати Web-сервіси, а також забезпечити захист додатків .NET Remoting на

основі потужних засобів захисту Microsoft Internet Information Services (IIS).

Відкритість .NET Remoting досягається за рахунок підтримки відкритих стандартів, таких, як HTTP, SOAP, Web Service Description Language (WSDL) та XML. Зв'язування закритих мереж за допомогою HTTP вирішується досі за допомогою тунелювання нестандартних мережевих форматів через HTTP, для чого розробляються спеціалізовані клієнти і сервери або використовуються Web-сервери. Підтримка стандарту WSDL потрібна для опису відкритих методів, параметрів і розташування Web-сервісів, а технологія Universal Description, Discovery, and Integration (UDDI) надає каталог для пошуку Web-сервісів.

Залежність серверного додатку JobServer від метаданих на клієнтському додатку JobClient полягає у тому, що конструктор класу JobServerlmpI у вигляді Web-сервісу підписується на повідомлення JobServerlmpI.JobEvent.

Конструктор класу JobServerlmpI може бути представлений у вигляді:

```
m_JobEventRepeater = new JobEventRepeater();
m_JobEventRepeater.JobEvent += new JobEventHandler (this.MyJobEventHandler);
m_IJobServer.JobEvent += new JobEventHandler (m_JobEventRepeater.Handler);
```

Для віддалених об'єктів, доступ до яких здійснюється за допомогою IIS, використовується канал HttpChannel, що обов'язково потрібно для Web-сервісів. Цей канал автоматично звертається до того ж порту, що IIS (за замовчуванням – порт 80) і містить жорстко зашитий URL сервера.

До кінцевої точки Web-сервісу можна звернутися в такий спосіб:

```
RemotingConfiguration.RegisterWellKnownClientType( typeof( SomeMBRType ),
"http://localhost/JobWebService/JobServer.soap?wsdl" );
```

Технологія .NET Remoting дозволяє писати Web-сервіси XML, які засновані на стандартах SOAP і HTTP або застосувати клас XmlSerializer для класів програми. Обмін повідомленнями на основі транспорту HTTP і SOAP застосовує схему "запит - відповідь". Клієнт посилає одне повідомлення – запит за певною адресою, у відповідь на яке сервер відповідає також одним повідомленням – відповіддю.

Для програмної установки режиму облікових даних клієнта необхідно за замовчуванням встановити властивість каналу в useDefaultCredentials при його створенні:

```
IDictionary props = new Hashtable( );
props["useDefaultCredentials"] = true;
HttpChannel channel = new HttpChannel(
        props,
        null,
        new SoapServerFormatterSinkProvider( )
);
```

Замість автоматичної передачі до сервера облікових даних користувача може використовуватися явна вказівка імені користувача, пароля та імені домена, що можливо тільки в поєднанні з будь-яким методом шифрування, наприклад SSL (Secure Sockets Layer).

Взаємозв'язок між екземпляром класу встановлюється для нового екземпляру JobEventRepeater і екземпляру JobServerlmpI. Таким чином, при генерації сервером події JobServerlmpl.JobEvent він викликає обробник в екземплярі класу JobEventRepeater.

Потрібний клас JobEventRepeater визначається наступним чином:

```
public class JobEventRepeater : MarshalByRefObject
{
        // Подія, на яку підписуються клієнти
        public event JobEventHandler JobEvent;
        // Оброблювач для JobServer.JobEvent
        public void Handler(object sender, JobEventArgs args)
        {
                if (JobEvent != null)
                {
                        JobEvent (sender, args);
                }
        }
        // Заборона на знищення об'єкта службами часу
        public override object InitializeLifetimeService( )
        {
                return null;
        }
}
```

Клас JobEventRepeate містить член JobEvent і метод RepeatEventHandter, який генерує подію JobEvent. Він також виступає в якості повторювача події JobEvent. Для того, щоб скористатися цим класом, клієнт створює його новий екземпляр і підписується на його подію JobEvent. Потім клієнт підписує даний екземпляр JobEventRepeater на подію JobSeiverlmpi.JobEvent, таким чином, що при генерації сервером події JobEvent він викликає метод JobEventRepeater.RepeatEventhandler.

Робоче завдання

1. Реалізувати інтерфейс користувача і обчислення заданої функції на сервері з використанням методів багатопоточності на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

2. Дослідити алгоритм виконання програми на мові C# або C++ та рівень забезпечення потокової безпеки при реалізації методів багатопоточності. Описати застосування базової моделі клієнт-сервер в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та реалізації коду на мові C# або C++ для авторизації користувача на Web-сервері необхідно використати конструктор мережевого класу TcpClientChannel для конфігурації каналів. У серверній частині програмного проекту реалізувати метод OnLogin() авторизації користувача на Web-сервері, наприклад:

```
void OnLogin(object sender, RoutedEventArgs e)
{
    try
    {
        labelValidatedInfo.Visibility = Visibility.Hidden;
        if (Membership.ValidateUser(textUsername.Text,
        textPassword.Password))
        {
```

```
                    // user validated!
                    labelValidatedInfo.Visibility = Visibility.Visible;
                }
                else {
                    MessageBox.Show("Username or password not valid",
                    "Client Authentication Services", MessageBoxButton.OK,
                    MessageBoxImage.Warning);
                }
            }
            catch (WebException ex) {
                MessageBox.Show(ex.Message, "Client Application Services",
                MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
```

2. У серверній частині програмного проекту створити методи ідентифікації користувача Initialize(), GetPassword(), ChangePassword(), ResetPassword(), UpdateUser(), DeleteUser(), ValidateUser() та інші, а також клас SampleMembershipProvider, який успадковує MembershipProvider та забезпечує доступ з клієнтів до Web-сервера через процедуру авторизації, наприклад:

```
namespace Wrox.ProCSharp.Security
{
    public class SampleMembershipProvider : MembershipProvider
    {
    private Dictionary<string, string> users = new Dictionary<string, string>();
    internal static string ManagerUserName = "Manager".ToLowerInvariant();
    internal static string EmployeeUserName = "Employee".ToLowerInvariant();
        public override void Initialize(string name, NameValueCollection config)
        {
            users.Add(ManagerUserName, "admin@mail");
            users.Add(EmployeeUserName, "user@mail");
            base.Initialize(name, config);
        }
```

```
public override string ApplicationName
{
        get { throw new NotImplementedException(); }
        set { throw new NotImplementedException(); }
}
public override bool ChangePassword(string username, string oldPassword,
string newPassword)
{
        throw new NotImplementedException();
}
public override bool DeleteUser(string username,
bool deleteAllRelatedData)
{
        throw new NotImplementedException();
}
public override bool EnablePasswordReset
{
        get { throw new NotImplementedException(); }
}
public override string GetPassword(string username, string answer)
{
        throw new NotImplementedException();
}
public override int MaxInvalidPasswordAttempts
{
        get { throw new NotImplementedException(); }
}
public override int PasswordAttemptWindow
{
        get { throw new NotImplementedException(); }
}
```

```
        public override string ResetPassword(string username, string answer)
        {
            throw new NotImplementedException();
        }
        public override bool UnlockUser(string userName)
        {
            throw new NotImplementedException();
        }
        public override void UpdateUser(MembershipUser user)
        {
            throw new NotImplementedException();
        }
        public override bool ValidateUser(string username, string password)
        {
            if (users.ContainsKey(username.ToLowerInvariant()))
            {
            return password.Equals(users[username.ToLowerInvariant()]);
            }
        return false;
        }
    }
}
```

3. За допомогою візуальних компонентів MFC і програмного додатку webBrowser дослідити процедуру авторизації користувача на Web-сервері в основу якої покладено модель клієнт-сервер, що реалізована в середовищі програмування MS Visual Studio 2010.

Контрольні питання

1. Які є способи розподілення та синхронізації процесів у мережі?
2. У чому полягає особливість процесів, що створені на основі моделі клієнт-сервер?
3. Поясніть використання конструктора мережевого класу TcpClientChannel.
4. Пояснити процедуру авторизації та роботу методів GetPassword() і ValidateUser().

3.4. Застосування технології .NET Remoting

Ціль роботи – застосування технології .NET Remoting при обчисленні заданої функції на сервері через реалізацію методів багатопоточності на основі базової моделі клієнт-сервер на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

Теоретичні відомості

При застосуванні технології .NET Remoting першим приймачем в ланцюжку канальних приймачів на клієнтській стороні є екземпляр клієнтського приймача, який реалізує інтерфейс ICIientFormatterSink. Клієнтський приймач виступає в якості моста між ланцюжком приймачів повідомлень і ланцюжком канальних приймачів. Таким чином, клієнтський приймач є приймачем повідомлень і канальним приймачем одночасно.

Нижче наведено приклад визначення нового класу MyClientFormatterSink, який використовує екземпляр MyFormatter клієнтського приймача:

```
public class MyClientForrnatterSink : CIientFormatterSink
{
        private IClientChannelSink _NextChannelSink;
        private  IMessageSink  _NextMessageSink;
        public MyClientFormatterSink( IClientChannelSink next )
        {
                _NextChannelSink = next;
        }
        public System.10.Stream GetRequestStream ( IMessage msg,
        ITransportHeaders headers )
        {
                // Цей приймач має бути першим у ланцюжку
                throw new NotSupportedException( );
        }
        public void ProcessMessage ( IMessage msg,
        ITransport Headers request Headers,
        Stream requestStream,
```

```
        out ITransportHeaders responseHeaders,
        out Stream responseStream )
        {
                // Цей приймач має бути першим у ланцюжку
                throw new NotSupportedException( );
        }
        public IMessage SyncProcessMessage ( IMessage ntsg )
        {
                // Поставити повідомлення в потік
                TransportHeaders requestHeaders =
                new TransportHeaders( );
                // Передати повідомлення по ланцюжку приймачів
                ITransportHeaders  responseHeaders = null;
                System.10.Stream responseStream =
                new System.10.MemoryStream ();
                // Оброблення об'єктів типу IMessage як типів MyMeasage
                MyHessage mr = (MyMessage)fm. Deserialize(responseStream);
                return mr.ConvertMyMessagePropertiesToMethodResponse(mc);
        }
}
```

Реалізований клієнтський приймач, який реалізує інтерфейс ICIientFormatterSink є першим приймачем в ланцюжку канальних приймачів. У загальному випадку реалізація SyncProcessMessage в клієнтському приймачі виконує функції:

1. отримує потік запиту для повідомлення запиту;
2. вбудовує повідомлення в потік запиту;
3. передає потік запиту до методу ProcessMessage для наступного канального приймача;
4. повертає повідомлення з потоку відповіді та обробляє його.

На відміну від клієнтського приймача, серверний приймач не є одночасно приймачем повідомлень і канальним приймачем, але лише останнім у ланцюжку. Серверний приймач реалізує інтерфейс IServerChannelSink та ставиться останнім у ланцюжку канальних

приймачів. Нижче визначено клас MyServerFormatterSink, що використовує розроблений раніше екземпляр MyFormatter клієнтського приймача:

```
public class MyServerFormatterSink : IServerChannelSink
{
    private IServerChannelSink _NextChannelSink;
    public MyServerFormatterSink{ IServerChannelSink snk)
    {
        _NextChannelSink = snk;
    }
    public ServerProcessing ProcessMessage(
    IServerChannelSinkStack sinkStack,
    IMessage requestMsg,
    ITransportHeaders request Headers,
    System.10.Stream requestStream,
    out IMessage responseMsg,
    out ITransportHeaders responseHeaders,
    out System. 10. Stream responseStream)
    {
        // Ініціалізація вихідних параметрів
        responseMsg  = null;
        responseHeaders = null;
        responseStream = null;
        // Підготовка до десеріалізації потоку запиту
        RemotingSurrogateSelector rem_ss = new RemotingSurrogate$elector();
        // Перетворити MyMessage в MethodCall.
        HethodCall me = mymsg. Convert MyMessagePropertiesToMethodCallf);
        msg = (IMessage)mc;
        // При виклику виконуючого (dispatch) приймача
        // потік запиту повинен бути null.
        ServerProcessing sp = this._NextChannelSink, ProcessMessage(
```

```
                    sinkStack,
                    msg,
                    requestHeaders,
                    null,
                    out responseMsg,
                    out responseHeaders,
                    out responseStream );
                return sp;
            }
        }
```

Реалізований серверний приймач, який реалізує інтерфейс IServerChannelSink є останнім приймачем в ланцюжку канальних приймачів. У загальному випадку серверний приймач виконує наступні дії:

1. десеріалізовує екземпляр з потоку запиту;
2. передає екземпляр методу ProcessMessage для наступного приймача в ланцюжку;
3. отримує потік відповіді для серіалізації відповідного повідомлення;
4. серіалізовує повідомлення, яке надіслано в потік відповіді.

Реалізація екземпляру ProcessMessage в MyServerFormatterSink визначає свої вихідні параметри, створює екземпляр класу MyFormatter і готує повідомлення до десеріалізації IMessage з потоку запиту. Якщо значення, що повертає ProcessMessage свідчить про те, що виклик методу завершений, отримуємо потік для серіалізації відповіді на запит, викликаючи методи об'єкту IServerChannelSinkStack. Це стандартна угода для отримання потоку, яке дозволяє приймачам в ланцюжку додавати інформацію в потік відповіді до того, як серверний приймач серіалізує відповідь.

Робоче завдання

1. Реалізувати інтерфейс користувача і обчислення заданої функції на сервері з використанням методів багатопоточності на мові C# або C++ в середовищі програмування Microsoft Visual Studio 2010.

2. Дослідити алгоритм виконання програми на мові C# або C++ та рівень забезпечення потокової безпеки при реалізації методів багатопоточності. Описати застосування базової

моделі клієнт-сервер в середовищі програмування Microsoft Visual Studio 2010.

3. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи та обчислення функції користувача на мові C# або C++ необхідно використати конструктор мережевого класу TcpClientChannel та методи конфігурації каналів передачі даних RegisterChannel і RemotingConfiguration для реалізації методів багатопоточності. У серверній частині програмного проекту додати необхідні директиви using для можливості звертання до класів системи .NET Remoting, наприклад:

```
// Простір імен, що використовується
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Runtime.Serialization.Formatters;
```

2. У серверній частині програмного проекту для передачі повідомлень між сервером і клієнтом потрібно створити і зареєструвати канал. Код для створення і налаштування каналу необхідно помістити в метод CreateChannel. При створенні каналу необхідно явно вказати можливість роботи з типами-делегатами, наприклад:

```
public class Form1 : System.Windows.Forms.Form
{
        TcpChannel CreateChannel(int port)
        {
                BinaryServerFormatterSinkProvider sp = new
                BinaryServerFormatterSinkProvider();
                // Дозволяємо передачу делегатів
                sp.TypeFilterLevel = TypeFilterLevel.Full;
                BinaryClientFormatterSinkProvider cp = new
                BinaryClientFormatterSinkProvider();
```

```
            IDictionary props = new Hashtable();
            props["port"] = port;
            return new TcpChannel(props, cp, sp);
        }
    }
```

3. У серверній частині програмного проекту в необхідно змінити код обробника пункту меню StartServer, який можна помістити в метод menuItem_Click. Для того, щоб об'єкти класів Game та Player могли бути доступні з інших комп'ютерів, ці класи необхідно зробити нащадками класу MarshalByRefObject, наприклад:

```
    private void menuItem2_Click(object sender, System.EventArgs e)
    {
        // Створюємо канал, який буде слухати порт 8000
        ChannelServices.RegisterChannel(CreateChannel(8000));
        // Створюємо об'єкт Game
        game = new Game();
        // Надаємо об'єкт Game для виклику з інших комп'ютерів
        RemotingServices.Marshal(game,"GameObject");
        // Підключення клієнта
        game.ShowPlayer += new ShowPlayerHandler(OnShowPlayer);
        player = game.Connect();
    }
    public class Player : MasrhalByRefObject
{
        // Реалізація класу Player
    }
    public class Game : MarshalByRefObject
{
        // Реалізація класу Game
    }
```

4. У клієнтській частині програмного проекту необхідно додати можливість підключення клієнта до сервера. Для цього потрібно до головного меню додати пункт

Connect to Server та до методу menuItem_Click помістити код створення каналу підключення до серверу і отримання посилання на об'єкт Game, наприклад:

```
private void menuItem4_Click(object sender, System.EventArgs e)
{
    string serverName = "localhost";  //  локальний комп’ютер
    // Створюємо канал, який буде підключений до сервера
    ChannelServices.RegisterChannel(CreateChannel(0));
    // Отримуємо посилання на об'єкт Game,
    // що розташований на іншому комп’ютері
    game = (Game)Activator.GetObject(typeof(Game),
    String.Format("tcp://{0}:8000/GameObject", serverName));
    // Підключення до серверу
    game.ShowPlayer += new ShowPlayerHandler(OnShowPlayer);
    player = game.Connect();
}
```

5. У клієнтській частині програмного проекту необхідно передбачити оновлення стану клієнтів при підключенні нових та відключенні діючих клієнтів. Для цього необхідно створити код методів Connect і Disconnect у класі Game, наприклад:

```
public class Game
{
    protected ArrayList players = new ArrayList();
    public event ShowPlayerHandler ShowPlayer;
    public Player Connect()
    {
        Player p = new Player(this);
        players.Add(p);
        OnShowPlayer(p,true);
        return p;
    }
    public void Disconnect(Player p)
```

```
        {
            OnShowPlayer(p,false);
            players.Remove(p);
        }
        public void OnShowPlayer(Player p,bool visible)
        {
            if(ShowPlayer != null)
            ShowPlayer(p,visible);
        }
    }
```

6. За допомогою візуальних компонентів MFC для програмних додатків дослідити алгоритм на основі моделі клієнт-сервер, що реалізований в середовищі програмування Microsoft Visual Studio 2010.

Контрольні питання

1. Які є способи розподілення та синхронізації процесів у мережі?
2. У чому полягає особливість процесів, що створені на основі моделі клієнт-сервер?
3. Поясніть використання конструктора мережевого класу TcpClientChannel.
4. Поясніть застосування технології .NET Remoting для синхронізації процесів.

4. Програмування цифрових сигнальних процесорів

Технологія цифрової передачі даних ефективно використовується у телемедицині, де відстань є критичним чинником. Телемедицина – це досить новий напрямок на стику декількох галузей медицини, телекомунікацій, інформаційних технологій. Безперечно, що одне з головних достоїнств телемедицини - це можливість надати висококваліфіковану допомогу фахівців провідних медичних центрів у віддалених районах і істотно заощадити при цьому витрати пацієнтів.

Телемедичні консультації здійснюються за допомогою передачі медичної інформації по електронних каналах зв'язку. Консультації можуть проводитися як в «відкладеному»

режимі по електронній пошті – найбільш дешевим і простим способом передачі медичної інформації, так і в режимі реального часу on-line з використанням каналів зв'язку і відеоапаратури. Планові і екстрені відеоконсультації і відеоконсиліуми – це безпосереднє спілкування між лікарем-консультантом та лікуючим лікарем, при необхідності – за участю хворого. Причому сеанс зв'язку відеоконференції може проходити як між двома абонентами, так і між декількома абонентами у так званому багатоточковому режимі, тобто найбільш складні випадки можуть обговорюватися консиліумом лікарів з різних медичних центрів.

Застосування мережевих відеокамер дозволяє організувати трансляцію хірургічної операції. Наприклад, через стандартні засоби Інтернет можна отримати доступ до відеокамер, встановлених у лабораторії телемедицини та операційних НАН України.

Отримують розвиток мобільні телемедичні комплекси (переносні, на базі реанімобіля і т.д.) для роботи на місцях аварій. Сучасний мобільний телемедичний комплекс повинен об'єднувати в собі потужний комп'ютер, легко сполучаються з різноманітним медичним обладнанням, засоби ближнього і далекого бездротового зв'язку, засоби відеоконференції і засоби IP-телефонії.

Телемедичні системи динамічного спостереження використовуються для спостереження за пацієнтами, що страждають хронічними захворюваннями. Ці ж системи можуть застосовуватися на промислових об'єктах для контролю стану здоров'я працівників (наприклад, операторів на атомних електростанціях). Новим напрямком розвитку дистанційного біомоніторингу є інтеграція датчиків в одяг, різні медичні аксесуари, мобільні телефони. Наприклад, існує жилет з набором біодатчиків, які реєструють ЕКГ, артеріальний тиск і ряд інших параметрів, або мобільний телефон з можливістю реєстрації ЕКГ і відправки їх засобами GPRS в медичний центр, а також з можливістю визначення координат людини в разі загрози життю пацієнта.

Доступність засобів зв'язку і сервісів Інтернет дозволяє розвивати такий напрямок, як «домашня телемедицина». Це дистанційне надання медичної допомоги пацієнту, що знаходиться поза медичної установи і проходить курс лікування в домашніх умовах. Спеціальне телемедичне обладнання здійснює збір і передачу медичних даних пацієнта з його будинку в віддалений телемедичний центр для подальшої обробки фахівцями. Прикладом може служити система моніторингу хворих із серцевою недостатністю, які

потребують регулярних і частих обстеженнях, вартість лікування яких істотно зменшується за рахунок використання телемедицини. Є комплекси, які включають датчики, що вимірюють температуру тіла, тиск крові, парціальний тиск кисню, ЕКГ і функції дихання, з'єднані з настільним монітором, який, в свою чергу, автоматично відправляє дані в контрольний центр. Крім того, можливий аудіовізуальний контакт з лікарями під час проведення консультації або діагностичної процедури.

Хоча сьогодні телемедицина залишається, в першу чергу, дистанційною діагностикою, її потенційні можливості значно ширше. До перспективних напрямків телемедицини відносяться телехірургія і дистанційне обстеження. Вони дозволяють проводити дистанційне керування медичною діагностичною апаратурою та дистанційні лікувальні впливи, хірургічні операції. В даний час деякі варіанти дистанційного керування вже входять в практику. Прикладом може служити управління мережевими відеокамерами, що є ефективним при спостереженні за станом пацієнтів в палатах інтенсивної терапії і дистанційному контролі хірургічних операцій. Іншим прикладом дистанційного управління є керування віддаленим мікроскопом, завдяки чому лікар-консультант отримує можливість проведення патогістологічного або патоцитологічного досліджень в повному обсязі, перегляду всіх наявних зразків матеріалу.

Проводиться експериментальне використання телекерованих маніпуляторів і дистанційне керування ними безпосередньо при проведенні операцій (управління скальпелем, лазером і т.д.). Найбільш відповідальний і складний з точки зору реалізації напрямок в даний час є суто експериментальні методики, впровадження яких в практику вимагає виконання багатьох технологічних інновацій.

Перспективи телемедицини пов'язані з подальшою мініатюризацією контрольно-вимірювальних засобів, впровадженням смарт-технологій, робототехніки, новітніх досягнень інформатики, прикладних аспектів нанотехнології.

У складі телемедичної системи (ТМС) можна виділити чотири типи елементів, взаємодія яких і утворює телемедичну мережу:

1. Каналоутворююче середовище – набір апаратних, програмних засобів, носіїв інформації і технологічних рішень (протоколи і стандарти), що забезпечують передачу різнорідної інформації в територіально розподіленому середовищі;

2. Консультаційний центр – медичний заклад, що має в штаті висококваліфікованих лікарів з різних напрямків медицини та відповідне обладнання для проведення дистанційних консультацій, консиліумів та лікувально-діагностичних процедур, а також організації навчання (проведення семінарів, лекцій) лікарів на віддалених станціях ТМС;

3. Диспетчерський пункт – виділена або функціонуюча у складі інших елементів ТМС структура, яка виконує функції фільтрації запитів на консультування, планування і забезпечення консультацій, організації консиліумів, а також збору і поширення інформації про можливості консультаційних центрів, а також містить службу адміністрування, що виконує функції супроводу мережевої структури;

4. Віддалені пункти – особливим чином обладнані медичні установи, персонал яких безпосередньо взаємодіє з пацієнтами і виконує комплекс лікувальних, діагностичних, профілактичних і реабілітаційних процедур.

При необхідності в структурі ТМС формуються тимчасові осередки – наприклад, комплекс віддалених медичних підрозділів в місцях бойових дій або техногенних катастроф. Такі станції розгортаються і підключаються до ТМС з метою залучення груп досвідчених фахівців провідних центрів до вирішення оперативних проблем, що виникають в таких місцях. Отримання консультацій пацієнтом можливе цілодобово за рахунок різниці в часі в різних часових поясах.

У структурі апаратного забезпечення телемедичних систем ви виділяється чотири основних складових: інфраструктура передачі мультимедійної інформації, комп'ютерне обладнання загального профілю, спеціалізоване комп'ютерне обладнання, спеціалізоване медичне обладнання.

Каналоутворююче середовище ТМС (інфраструктура передачі мультимедійної інформації) не залежить від носія інформації – це можуть бути кабельні провідні структури, волоконно-оптичні канали та канали супутникового і радіозв'язку. Устаткування і канали зв’язку забезпечують передачу різнорідної інформації – алфавітно-цифровий і графічної, аудіо та відео, а також цифрових і аналогових сигналів, що знімаються з датчиків, і даних, що передаються на органи управління діагностичної лікувальної апаратури. Кінцеве обладнання забезпечує перетворення і узгодження сигналів, їх перекодування з одного формату в інший, а також здійснює їх компресію/декомпресію. Слід зазначити, що сучасні

ТМС для зв’язку на відеоконференціях можуть ефективно працювати в різних мережевих токологіях, побудованих на основі протоколів IP, ISDN, ATM і ін. У якості служб надання сервісів виступають розподілені сервери програмних додатків і архівації. Організація багатоточкового відеозв’язку, ведення розкладів консультацій і сервісів дистанційного навчання і тестування виконується на серверах програмних додатків. Служби архівації забезпечують довгострокове зберігання великих обсягів інформації, їх каталогізацію і пошук. Комп'ютерне обладнання загального профілю служить для організації робочих місць лікаря-консультанта і лікаря, пультів централізованого моніторингу, а також для обладнання операційних. До його складу входять комп'ютери різної архітектури і призначення. Крім комп'ютерів сюди входить різне периферійне устаткування – кодеки відеозв’язку, відеокамери, аудіосистеми, різні дигітайзери.

Склад спеціалізованої комп'ютерного обладнання визначається виходячи з потреб конкретних медичних програм і може містити спеціалізовані сканери, пристрої управління, спеціалізовані системи відображення відеографічної інформації, а також пристрою сполучення комп'ютерного та спеціалізованого медичного обладнання.

Діагностичне, лікувальний і реабілітаційне обладнання може підключатися до ТМС безпосередньо і через пристрої сполучення. При неможливості або недоцільність такого підключення інформація з такого обладнання може перетворюватися в цифрову форму з використанням спеціального обладнання – сканерів, дигітайзерів і т.п.

Захист зберігаємої і переданої інформації, авторизація доступу до ТМС, і нарешті, забезпечення живучості мережі в різних режимах функціонування (мирний час, надзвичайні події тощо) утворює комплекс програмно-апаратних засобів та управлінческих рішень системи безпеки ТМС. Для забезпечення захисту інформації, що зберігається в архівах і передається по каналах зв'язку, використовуються апаратні і програмні криптографічні засоби.

Авторизація доступу лікарів до обладнання ТМС актуальна як при проведенні телеконсультацій для підтвердження повноважень спеціалістів, так при роботі з терміналами для запобігання несанкціонного доступу до медичних даних. Засоби електронного підпису використовуються для верифікації документів, які реєструють результати телеконсультацій, віддаленого тестування і т.п.

4.1. Технологія передачі даних RTDX

Ціль роботи – проектування цифрового фільтру для монітору пацієнта і медичного вимірювача у LabVIEW з мікропроцесорним управлінням і дослідження його статичних і динамічних параметрів.

Теоретичні відомості

Один з варіантів побудови віддаленого діагностичного пункту на основі мікропроцесора наведено на рис. 4.1.1.

Системні рішення для малопотужних діагностичних центрів включають такі ключові компоненти:

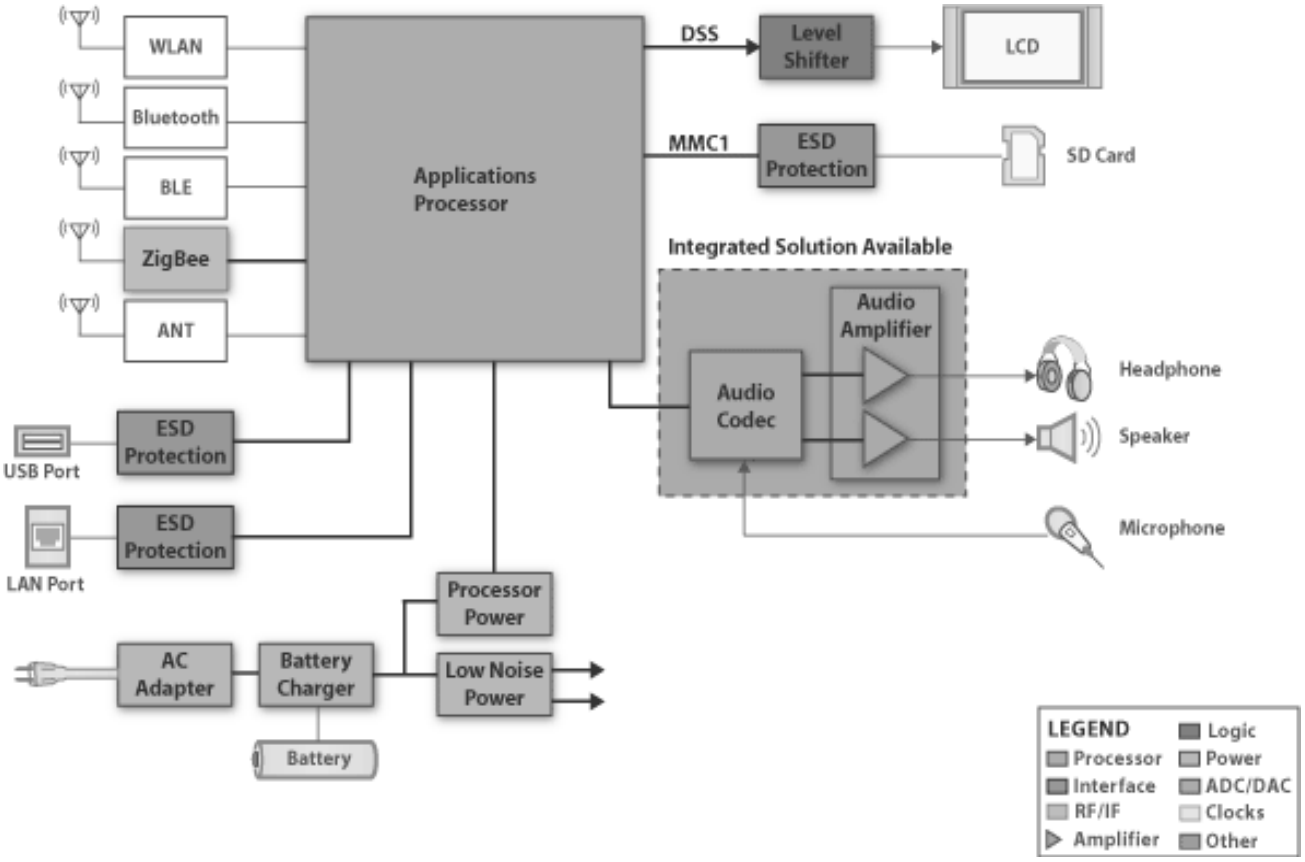


Рис. 4.1.1. Структурна схема діагностичного пункту

1. Мікропроцесор. У якості мікропроцесора використовується LM3S9B9x, побудований на базі ARM Cortex M3 з вбудованими Ethernet і високошвидкісним USB-

трансівером. Низьке енергозабезпечення в режимі сну і швидка обробка переривань є ідеальними можливостями для подібних пристроїв, що використовують в якості джерела живлення акумуляторних батарей. Виробник мікропроцесорів надає всі необхідні системні драйвери, тому ніякої спеціальної програмної збірки не потрібно.

2. Зв'язок. Для того щоб пристрій було сертифіковано, він повинен включати, принаймні, один Personal Area Network (PAN) інтерфейс, такий як USB і Bluetooth, і один інтерфейс Wide Area Network (WAN) для локальної мережі (LAN), такий як Ethernet і ZigBee.

3. Аудіо. Інтегровані роз’єми для навушників і підсилювач гучномовця забезпечує низьке енергоспоживання, низьку вартість і малі розміри.

4. Керування живленням. У пристрої застосовується спеці ний блок живлення і процесор, які забезпечують низьку вартість, низький рівень шуму, низький струм спокою для сигнальних ланцюгів і процесора. А одноелементний літій-іонний акумулятор може бути використаний тривалий час.

Для використання в телемедичних мережах оптимально підходить спеціалізоване медичне обладнання, що має візуальний або акустичний зворотній зв'язок з лікарем, а також вбудовану мережеву підтримку. Для кардіології це можуть бути ангіографічне установки і різні ехографи, в пульмонології – це бронхоскоп, в гастроентерології – гастроскопи, в дерматології і ендоскопії – дерматоскопи і відеокамери з ендоскопічними насадками. Також це може бути діагностичне обладнання широкого профілю – апарати для ультразвукового дослідження, ЯМР-томографи, мікроскопи, стетоскопи та інше обладнання.

Робоче завдання

1. Зібрати досліджувану схему віртуального приладу в National Instruments LabVIEW 2010 (рис 4.1.2 і рис. 4.1.3).

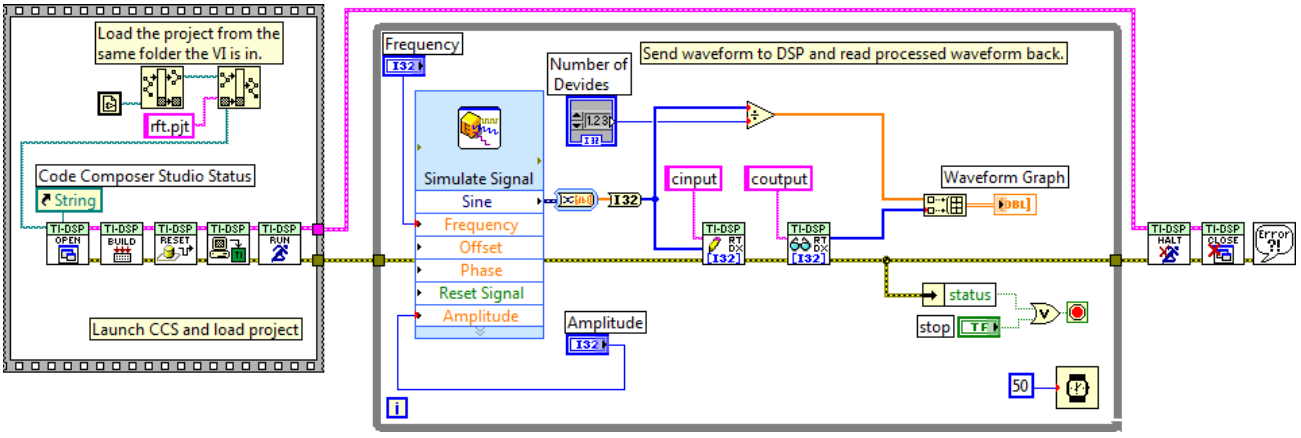


Рис. 4.1.2 Схема віртуального приладу в National Instruments LabVIEW 2010

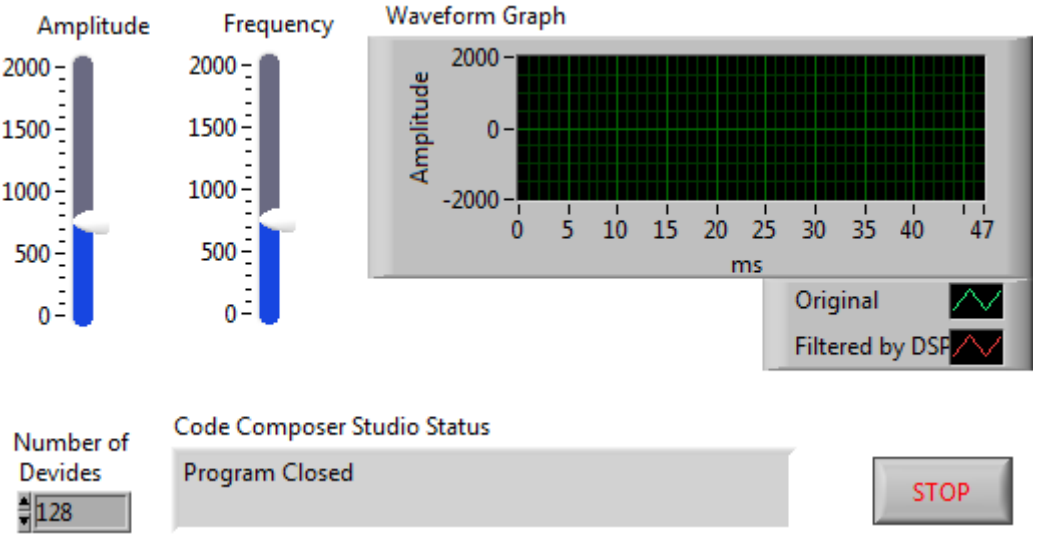


Рис. 4.1.3 Зовнішній вигляд системи керування приладом

2. Дослідити роботу компонентів LabVIEW DSP Test Integration Toolkit. Описати призначення і функціональні можливості компонентів низького рівня RTDX та CCS IDE для віртуального інтерфейсу DSP Test Integration Advanced VI.

3. Задокументувати для звіту зняті в експериментах частотні діаграми і характеристики. Сформулювати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи необхідно використати елементи

керування National Instruments LabVIEW 2010, що представлені на рис. 4.1.2 і 4.1.3. Установити задані викладачем параметри фільтру:

Amplitude (A) = 0.50V;

Frequency (F) = 1e+03Hz;

2. Задіяти програмні методи управління цифровим фільтром за допомогою компонентів DSP Test Integration Advanced VI на рівні RTDX та CCS IDE при розробці програмного забезпечення в середовищі програмування мікропроцесорів Code Composer Studio:

```
// Ініціалізація змінних і каналів RTDX
        #define BUFFER_SIZE 48
        RTDX_CreateInputChannel(cinput);
        RTDX_CreateOutputChannel(coutput);
        int input[BUFFER_SIZE], output[BUFFER_SIZE];
// Активація каналів RTDX
        RTDX_enableInput(&cinput);
        RTDX_enableOutput(&coutput);
// Читання-запис у канали RTDX
        RTDX_read(&cinput, input, sizeof(input));
        RTDX_write(&coutput, &output, sizeof(output));
```

3. Реалізувати цифровий фільтр (FIR Filter) з можливістю програмного регулювання коефіцієнтів фільтра при розробці програмного забезпечення в середовищі програмування мікропроцесорів Code Composer Studio:

```
// The Simple FIR Filter
double FIRFilter (double val, int nTaps, double* history, double* coefs)
{
    double temp, filtered_val, hist_elt;
    int       i;
    hist_elt = val;
    filtered_val = 0.0;
    for (i = 0; i<nTaps; i++)
```

```
    {
        temp = history[i];
        filtered_val += hist_elt * coefs[i];
        history[i] = hist_elt;
        hist_elt = temp;
    }
    return filtered_val;
}
```

4. Завантажити програмне забезпечення для мікроконтролеру Texas Instruments DSK6400, що розроблено в середовищі програмування Code Composer Studio і компільоване у проект:

Project (PJT) – example.pjt.

За допомогою графічних засобів Waveform Graph отримати частотні діаграми і характеристики цифрового фільтру.

Контрольні питання

1. Що таке компоненти DSP Test Integration Advanced VI?
2. З яких міркувань вибрано мікроконтролер Texas Instruments DSK6400?
3. Поясніть принципи керування каналами передачі даних на рівні RTDX та CCS IDE.
4. Поясніть процедуру ініціалізації змінних і каналів RTDX.
5. Від яких параметрів залежить активація каналів RTDX.
6. Поясніть роботу компонентів CCS Build Project і CCS Download Code.

4.2. Передача даних у телемедичних системах

Ціль роботи – проектування цифрових фільтрів ФНЧ і ФВЧ в каналах вимірювання для телемедичної системи з мікропроцесорним управлінням через вбудований Ethernet і високошвидкісний USB-трансівер.

Теоретичні відомості

На малюнку 4.2.1 представлений варіант використання мікроконтролеру для створення телемедичного консультаційного центру з відеовиходом.

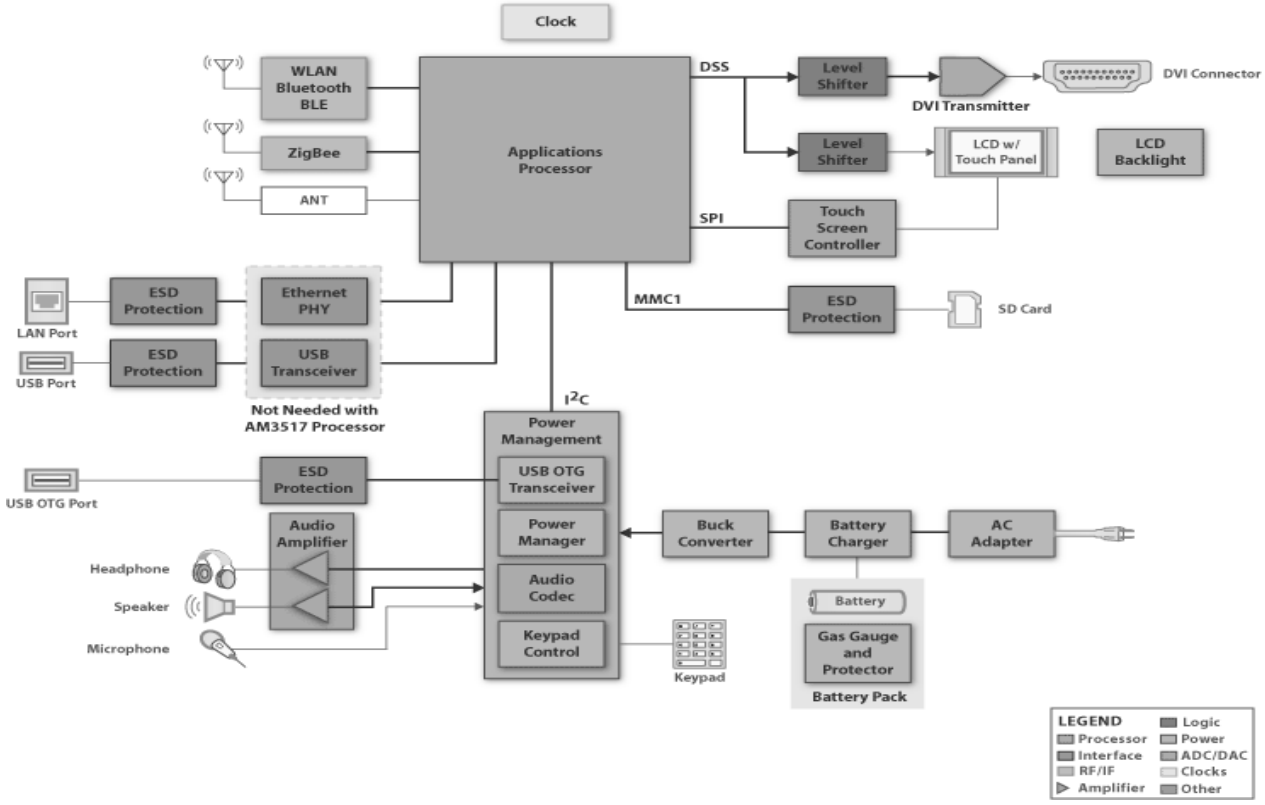


Рис. 4.2.1. Структурна схема телемедичного консультаційного центру

Такі консультаційні центри повинні забезпечувати відмінні зображення і звук. Тому системні рішення включають такі ключові компоненти:

1. Процесор. Процесор OMAP3530 має ARM Cortex A8 орієнтовану архітектуру, яка забезпечує аудіо- та відеоінтерфейси периферійних пристроїв. Вона включає в себе HD-відеоприскорювач для включення потокового відео високої роздільної здатності. Для TMC систем, які не потребують потокового HD-відео, слід використовувати AM3517, який не підтримує HD-відеоприскорювач. Обидва процесора входять в останні дистрибутиви Linux і Windows сумісні за кодом.

2. Зв'язок. Для того щоб пристрій було сертифіковано, він повинен включати, принаймні, один Personal Area Network (PAN) інтерфейс, такий як USB і Bluetooth, і один інтерфейс Wide Area Network (WAN) для локальної мережі (LAN), такий як Ethernet і ZigBee.

3. Керування живленням. Процесор розроблений для роботи разом з OMAP35x-пристроями. Він містить перетворювач, низькорівневий регулятор, зарядний модуль, звуковий модуль з цифровим фільтром, вхідні і вихідні підсилювачі класу D. Також є кілька додаткових функцій, таких як високошвидкісний USB-трансивер.

4. Управління батареєю. Для того щоб підтримувати потокове відео, бездротовий зв'язок і великий екран, необхідні батареї великої ємності з декількома елементами, з'єднаними послідовно і паралельно.

5. DVI. TFP410-DVI 1.0, сумісний цифровий передавач, який підтримує роздільну здатність дисплею, починаючи від VGA до UXGA в 24 біта. Деякі з переваг цього універсального інтерфейсу включають вибір розрядності шини, рівні сигналу, що налаштовуються, а також диференційний і несиметричний розгін відеопотоку.

Доступ до ресурсів телемедичних систем з зовнішніх мереж зв'язку забезпечується використанням програмних систем – брандмауерів. Живучість ТМС забезпечується як топологією ТМС, що має структуру дублюючих каналів різної фізичної природи і інтелектуальних комутаторів, так і заходами по розподіленому архівному зберіганні інформації.

Робоче завдання

1. Зібрати досліджувану схему віртуального приладу в National Instruments LabVIEW 2010 (рис 4.2.2 і рис. 4.2.3).

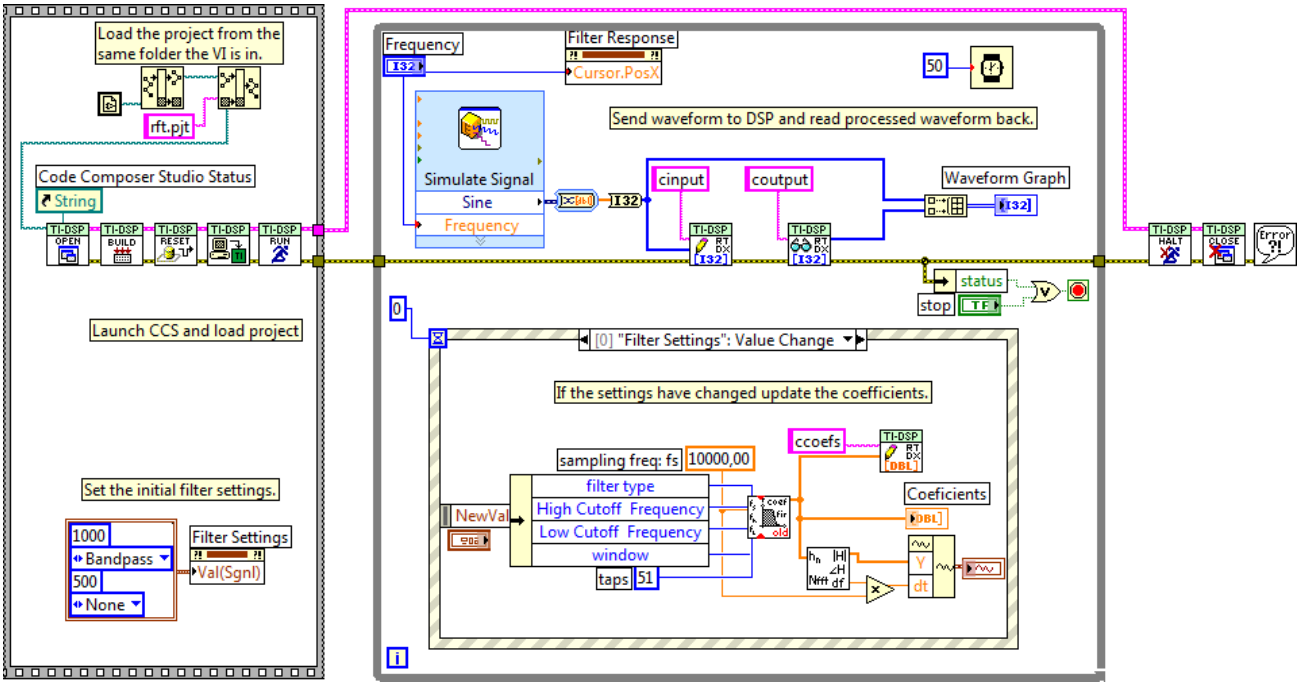


Рис. 4.2.2 Схема віртуального приладу в National Instruments LabVIEW 2010

2. Дослідити роботу компонентів LabVIEW DSP Test Integration Toolkit. Описати призначення і функціональні можливості компонентів низького рівня RTDX та CCS IDE для віртуального інтерфейсу DSP Test Integration Advanced VI.

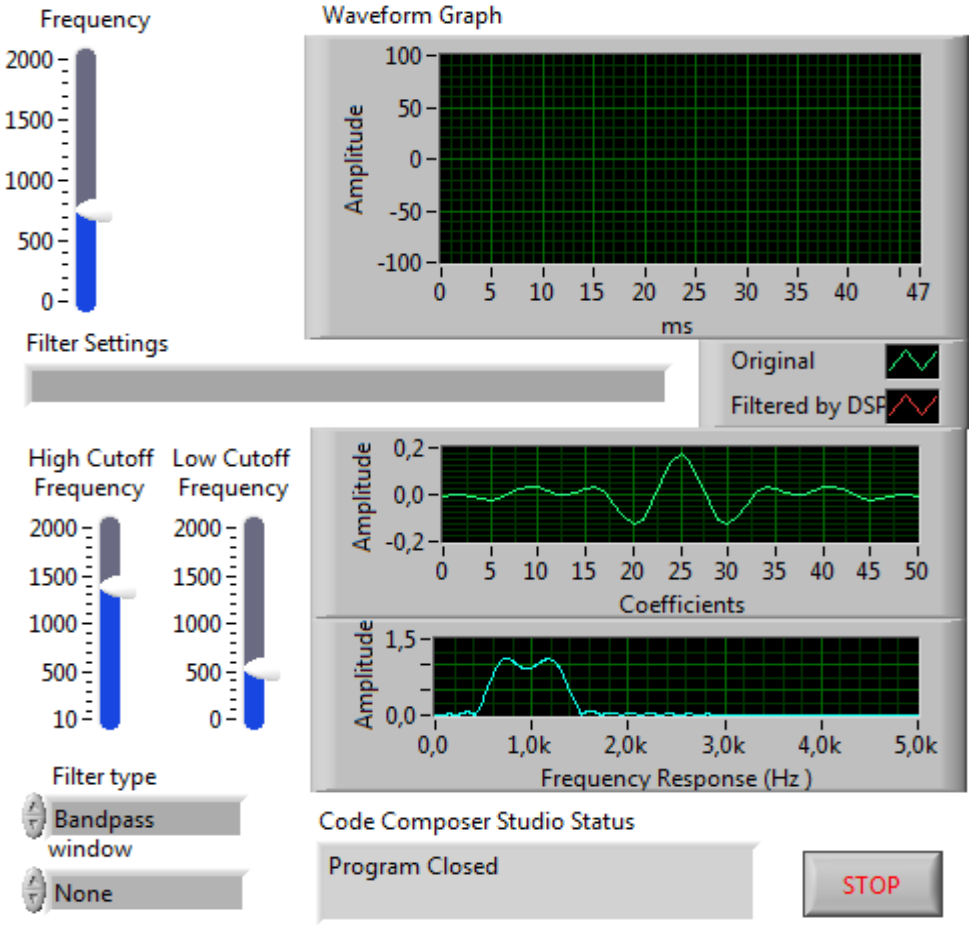


Рис. 4.2.3 Зовнішній вигляд системи керування приладом

3. Задокументувати для звіту зняті в експериментах частотні діаграми і характеристики. Сформулювати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

1. Для виконання даної лабораторної роботи необхідно використати елементи керування National Instruments LabVIEW 2010, що представлені на рис. 4.2.2 і 4.2.3. Установити задані викладачем параметри фільтру:

Amplitude (A) = 0.50V;

Low Cut-off Frequency (LF) = 500Hz;

High Cut-off Frequency (HF) = 1e+03Hz;

2. Задіяти програмні методи управління цифровим фільтром за допомогою компонентів DSP Test Integration Advanced VI на рівні RTDX та CCS IDE при розробці програмного забезпечення в середовищі програмування мікропроцесорів Code Composer Studio:

```
// Ініціалізація змінних і каналів RTDX
        #define BUFFER_SIZE 48
        RTDX_CreateInputChannel(cinput);
        RTDX_CreateInputChannel(ccoefs);
        RTDX_CreateOutputChannel(coutput);
        int input[BUFFER_SIZE], output[BUFFER_SIZE];
// Активація каналів RTDX
        RTDX_enableInput(&cinput);
        RTDX_enableInput(&ccoefs);
        RTDX_enableOutput(&coutput);
```

3. Реалізувати читання / запис даних з цифрових каналів за допомогою компонентів DSP Test Integration Advanced VI на рівні RTDX та CCS IDE в середовищі програмування мікропроцесорів Code Composer Studio:

```
int i;
double filtered;
// Читання-запис у канали RTDX
for ( ; ; )
{
    /* Wait for new buffer */
    while(!RTDX_read(&cinput, input, sizeof(input)));
    /* See if there are new coefficients to be read */
    if (!RTDX_channelBusy(&ccoefs))
    RTDX_readNB(&ccoefs, &gFIRCoefficients, sizeof(gFIRCoefficients));
    for(i=0; i<BUFFER_SIZE; i++)
    {
    filtered = BandpassFilter (input[i]*gain, kTAPS, gFIRHistory, gFIRCoefficients);
```

```
        output[i] = (int)(filtered + 0.5); // round to nearest
        }
        RTDX_write(&coutput, &output, sizeof(output));
}
```

4. Реалізувати цифровий смуговий фільтр ФНЧ і ФВЧ в середовищі програмування мікропроцесорів Code Composer Studio:

```
// The simple Band-Pass Filter
double BandpassFilter (double val, int nTaps, double* history, double* coefs)
{
double temp, filtered_val, hist_elt;
int i;
hist_elt = val;
filtered_val = 0.0;
        for (i = 0; i<nTaps; i++)
        {
        temp = history[i];
        filtered_val += hist_elt * coefs[i];
        history[i] = hist_elt;
        hist_elt = temp;
        }
return filtered_val;
}
```

Або реалізувати цифровий смуговий фільтр Бесселя заданого викладачем порядку фільтра (n) в середовищі програмування мікропроцесорів Code Composer Studio, коефіцієнти якого (a, b) розраховуються студентом самостійно:

```
// Bessel Filter Coefficients for n = 6
int npoints, index, j, k1, k2, xr, tr;
int n, n_index, n_dat, n_teil, n_fil, data;
float daten[256], koef[4][3], a[4][3], b[4][3];
float spei_x[4][3], spei_y[4][3], x[256], y[256], xtime[256];
```

```
void BesselFilter (void)
{
        for (j=1;j<a_file;j++)
        {
        xtime[j]= j/(a_file);
        x[j]= exp(-cos(j)/sin(a_file))+exp(cos(j)/sin(a_file));
        }
/* n=6, F1=48Hz */
n = 6;
/* F2=1000Hz */
/* Coefficient a: */
a[1][0]=0.927303;  a[1][1]=1.854607;  a[1][2]=0.927303;
a[2][0]=0.941331;  a[2][1]=1.882663;  a[2][2]=0.941331;
a[3][0]=0.967707;  a[3][1]=1.935413;  a[3][2]=0.967707;
/* Coefficient b: */
b[1][0]=1.0;      b[1][1]=1.851753;   b[1][2]=0.857460;
b[2][0]=1.0;      b[2][1]=1.880051;   b[2][2]=0.885275;
b[3][0]=1.0;      b[3][1]=1.933302;   b[3][2]=0.937524;
for(n_teil=1;n_teil<(int)(n|2);n_teil++)
{
        for(n_fil=1;n_fil<2;n_fil++)
        {
        spei_x[n_teil][n_fil]=0;
        spei_y[n_teil][n_fil]=0;
        }
}
for(n_dat=1;n_dat<a_file;n_dat++)
{
        spei_x[1][0]=x[n_dat];
        for(n_teil=1;n_teil<(int)(n|2);n_teil++)
```

```
{
spei_y[n_teil][0]=spei_x[n_teil][0]*a[n_teil][0]+
        spei_x[n_teil][1]*a[n_teil][1]+
        spei_x[n_teil][2]*a[n_teil][2];
spei_y[n_teil][0]=spei_y[n_teil][0]*b[n_teil][0]-
                spei_y[n_teil][1]*b[n_teil][1]-
                spei_y[n_teil][2]*b[n_teil][2];
spei_x[n_teil][2]=spei_x[n_teil][1];
spei_x[n_teil][1]=spei_x[n_teil][0];
if(n_teil<(int)(n|2))
{
spei_x[n_teil][0]=-spei_y[n_teil][0]; }
spei_y[n_teil][2]=spei_y[n_teil][1];
spei_y[n_teil][1]=spei_y[n_teil][0];
}
y[n_dat]=spei_y[1][0];
}
for(n_index=1;n_index<a_file;n_index++)
{
 x[n_index]=y[n_index-1];
}
for(index=30;index<a_file;index++)
{
time[index-30], x[index];
}
}
```

5. Завантажити програмне забезпечення для мікроконтролеру Texas Instruments DSK6400, що розроблено в середовищі програмування Code Composer Studio і компільоване у проект:

Project (PJT) – example.pjt.

За допомогою графічних засобів Waveform Graph отримати частотні діаграми і характеристики цифрового фільтру.

Контрольні питання

1. Які типи елементів можна виділити у складі телемедичної системи?
2. Що таке компоненти DSP Test Integration Advanced VI?
3. З яких міркувань вибрано мікроконтролер Texas Instruments DSK6400?
4. Поясніть принципи керування каналами передачі даних на рівні RTDX та CCS IDE.
5. Поясніть процедуру ініціалізації змінних і каналів RTDX.
6. Поясніть роботу компоненту CCS RTDX Read Array DBL.

Література

1. Таненбаум Э. Распределенные системы: принципы и прадигмы / Э. Таненбаум. – Питер, 2003. – 977 стр.

2. Дейтел Х. М. Операционные системы. Распределенные системы, сети, безопасность: учеб. пособие / Х. М. Дейтел – 3-е изд. – Бином-пресс, 2011. – 704 с.

3. Морган С. Разработка распределенных приложений на платформе Microsoft .Net Framework / С. Морган, Б. Райан, Ш. Хорн, М. Бломсма, 2008. – 608 с.

4. Tanenbaum A. Computer Networks. Englewood Cliffs / A. Tanenbaum. – NJ: Prentice Hall, 3rded., 1996.

5. Федоров А. Windows Azure. Облачная платформа Microsoft/ А. Федоров, Д. Мартынов, – N-Y.: Microsoft Press, 2010. – 100 с.

6. Фаулер. Архитектура корпоративных программных приложений / Фаулер, Мартин. – М.: Издательский дом «Вильяме», 2006. – 544 с.

7. Фостер Я. Анатомия ГРИД. Создание Масштабируемых виртуальных организаций / Я. Фостер, К. Кессельман, С. Тьюк. – 2003.

8. Фостер Я. Физиология ГРИД. Открытая архитектура грид-служб для интеграции распределённых систем / Я. Фостер, К. Кессельман, Д. Ник, С. Тьюк. – 2003.

9. Пухальский, Г.И. Проектирование микропроцессорных устройств : учебное пособие для вузов / Г.И. Пухальский. – СПб. : Политехника, 2001. – 588 с.

10. www.ti.com